CDC 00594

The Transactor

The Tech/News Journal For Commodore Computers

95% Advertising Free! Sept. 1986: Volume 7, Issue 02. $3.50

GAMES

"From the Inside Out"

Good news!

If you want to get the most out of your Commodore 128 or 64, we have goods news for you. The Pocket 128 and 64 Series of Software both offer you serious, professional quality software packages that are easy to use and inexpensive.

How easy?

Pocket 128 or 64 Software is so easy, you're ready to start using it as soon as it's loaded into memory. Even if you've never been in front of a computer before, you'll be up and running in thirty minutes. In fact, you probably won't ever need the reference guide . . . 'help' is available at the touch of a key. That's how easy.

How serious?

Pocket 128 or 64 packages have all the power you're ever likely to need. They have all of the features you'd expect in top-of-the-line software, and then some. The good news is that Pocket 128 or 64 Software Packages are priced way down there . . . where you can afford them.
Fast, powerful, easy to learn and inexpensive.
Say, that is good news!

All for one and one for all

Pocket 128 or 64 Software Packages offer you something else you might not expect . . . integration. You can combine the output of Pocket Writer, Pocket Filer and Pocket Planner into one piece of work. You can create a finished document with graphs, then send individually addressed copies.

The bottom line is Solutions

The word solutions is our middle name and bottom line. When you purchase Pocket 128 or 64 software, you can count on it to solve your problems.

For information write to:
Digital Solutions
30 Wertheim Court, Unit 2
Richmond Hill, Ontario
Canada L4B 1B9
telephone (416) 731-8775

™PaperClip is a registered trademark of Batteries Included

™Visicalc is a registered trademark of Software Arts

Pocket Writer 128 or 64 Word Processing

What you see is what you get

With Pocket Writer 128 or 64, there's no more guessing what text will look like when you print it. What you see is what you get . . . on screen and in print. There are no fancy codes to memorize, no broken words at the end of a line.

Easy to learn and sophisticated. Pocket Writer 128 or 64 offers standard word processing features plus . . .

- on-screen formatting and wordwrap
- on-screen **boldface**, underlines and *italics*
- no complicated format commands to clutter text
- on-screen help at all levels
- spelling-checker lets you add words to your dictionary
- 40 or 80 columns on screen
- files compatible with PaperClip™ or other word processors

Pocket Planner 128 or 64 Computerized Spreadsheet

Make fast work of budgeting and forecasting

Pocket Planner 128 or 64 software lets you make fast work of all your bookkeeping chores. Cheque books, household accounts, business forecasting and bookkeeping are just some of the jobs that Pocket Planner 128 or 64 packages make easier. You can even create four different kinds of graphs.

Accurate, sophisticated and easy to use. Pocket Planner 128 or 64 offers standard spreadsheet features plus . . .

- accuracy up to 16 digits, about twice as many as most spreadsheets for the Commodore 128 or 64
- sideways printing available on dot matrix printers, for oversized spreadsheets that won't fit on standard paper
- **on-screen help** at all levels
- compatible with VisiCalc™ files
- 80 column on-screen option for the Commodore 64 in addition to the standard 40 columns
- graphics include **bar**, **stacked bar**, **line** and **pie** graphs that can also be used in word processing files
- **smart evaluation** of formulae for accurate complex matrices

Pocket Filer 128 or 64 Database Manager

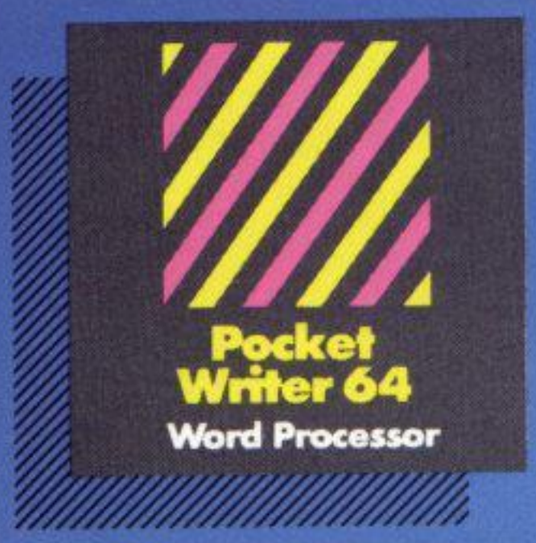
Database management made easy

With Pocket Filer 128 or 64, you can organize mailing lists, addresses, inventories, telephone numbers, recipes and other information in an easily accessible form. Use it with Pocket Writer 128 or 64 (or other word processors) to construct individually customized form letters.

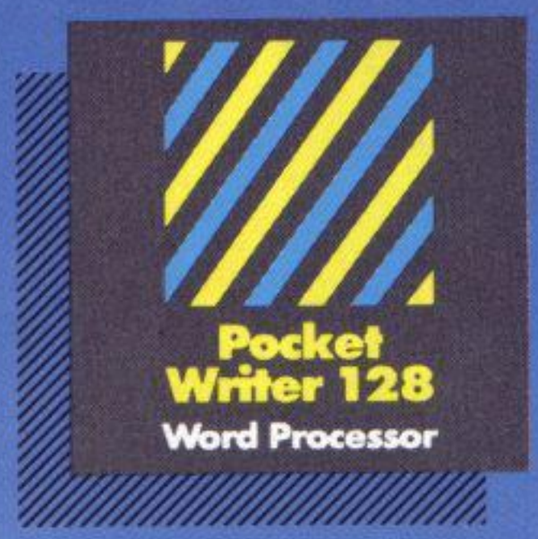
Pocket Filer 128 or 64 packages are fast, sophisticated and truly easy to use. In addition to standard database features they offer . . .

- use up to 255 fields per record (2,000 characters per record)
- sorts by up to 9 criteria, can save 9 different sorts
- print **labels** in multiple columns
- flexible report formatting including **headers** and **footers**
- optional password protection including **limited access viewing** or **updating**
- on-screen help at all levels
- print from any record to any record
- arithmetic and trigonometric functions in **reports** using up to 16 digit accuracy

Solutions!



PW 128/64 Dictionary
also available at $14.95 (U.S.)



MAIL ORDERS:
Transactor Publishing Inc.
500 Steeles Avenue
Milton, Ontario, L9T 3P7
1-416-878-8438
Or use order card at center.

Only The Name Is New

The professional, full-featured software line from Digital Solutions is now called Pocket Software.
Pocket Writer 128/64.
Pocket Filer 128/64.
Pocket Planner 128/64.
The names are new, but this super software is still the same.

From now on, when you hear the word Pocket, it means software that's full-featured, handy and easy to use.

Pocket Software at prices that won't pick your pocket.

Best-selling software for Your Commodore 128 or 64

You want the very best software you can find for your Commodore 128 or 64, right?

You want integrated software — word processing, database and spreadsheet applications — at a sensible price. But, you also want top-of-the-line features. Well, our Pocket 128/64 software goes one better.

With Pocket 128 or 64, you'll find all the features you can imagine . . . and then some. And Pocket 128/64 is so easy to use, you won't even need the reference guide. On-screen and in memory instructions will have you up and running in less than 30 minutes, even if you've never used a computer before.

The price? It's as low as you'd expect for a line of software called 'Pocket'. Suggested Retail Price for the 64 software is $39.95 (U.S.) and $49.95 (U.S.) for the 128. Any of the 64 products may be upgraded to their 128 version for $15.00 (U.S.) + $3.00 shipping and handling. (Available to registered owners from Digital Solutions Inc. only.)

Pocket Writer 128 or 64, Pocket Planner 128 or 64 and Pocket Filer 128 or 64 . . . **Solutions** at sensible prices from Digital Solutions Inc.

International & Distributor enquiries to:

Digital Solutions Inc.

30 Wertheim Court, Unit 2
Richmond Hill, Ontario
Canada L4B 1B9
telephone (416) 731-8775

Serious software that's simple to use.

Pocket Writer 128 and 64 are now available in French.

«ЭЛЕКТРОНИКА»
«ИНФОРМАТИКИ»
г. Ужгород, ул. Замковая, 4/7
тел. 3-68-83

2 MEGs For Your AMIGA

A must for software developers
Allows more programs to run simultaneously and faster
Can be used to increase system RAM and/or as a FAST RAM DRIVE
Uses standard memory bus architecture to allow for future compatibility
Allows full use of memory expansion port for additional peripherals

AX2000 2 MEG RAM Board $899.00 U.S. ($1276.00 CDN)
AX1000 1 MEG RAM Board $729.00 U.S. ($1035.00 CDN)

Complete in case, nothing else to buy!
1 year manufacturer warranty!

DEALER INQUIRIES INVITED

Comspec Communications Inc.
153 Bridgeland Avenue, Unit 5
Toronto, Ontario, Canada
M6A 2Y6 (416) 787–0617

Mail order through The Transactor
(see order card and News BRK)

Shipping via courrier: within Canada add $25.00. To U.S.A. add $100.00 U.S. - includes customs clearance

AMIGA is a registered trademark of Commodore Business Machine.

Volume 7
Issue 02
Circulation 64,000

Games From The Inside Out

Start Address Editorial 3

Bits and Pieces 5
Head Cleaning For The 2031
SuperGET
Turning off REMOTE 64
Freakout and Farzoid
Device Not Present Update
Display T&S Fix
Underline Cursor for the 64
Complete Joystick Keyboard Equivalents
Define Extra C-128 Keys
Setting the B-128 Interrupt Rate
More B-128 Info
B-Series Sound
Find A Program's Start and End Address
Finding Relative File Record Lengths
SID Whistle-Killer
Disk Rescue
LOAD By Track and Sector
A Use For The Periodic Table!
Pythagorean Triplet Generator
Easy DATA Viewing
Textcraft Files On Amiga's External Drive
C-64 Easy Filename Retrieval
C128 CP/M ASCII Output
Un-Assembler Speed-Up
Bigger Stacks For Healthy Assembly
Amiga ABasiC Program Boot
Easy Speed-Up For The 1541
Amiga Date-Prompt Startup Sequence

Letters 13
The Mutant Vic
The Copy Blues
Postal Service Blues
Vic Time
Hex - An Evil Spell
1200 Bits Per Second per second
Left Wing Interference Revisited #2
A Comparison Of Four Word Processors Revisited

News BRK 76
Amiga RAM Expansion by Comspec
Paperback Writer now "Pocket Writer"
Gnome Speed Compiler = SM Compiler
No Sales Tax on Books
Sending Cheques For Transactor Products
The Transactor Communications Disk
Demise of Viewtron
Quantum Link and Timeline
Using Transactor Programs In Proprietary Software
1986 Midwest Commodore Conference/Expo
MSD Still Alive And Prospering!
Twin Cities 128: The Commodore 128 Journal
Creative Writer
Sector Surgeon For The C-64
Three MIDI Data Storage Programs For The 64
The Electronic Shoe Box Accounting Systems
Freedom Assembler-128 For The Commodore 128
Rebel Assembler/Editor For The 64 and 128
Liz Deal's Basic Program Converter
10 and 20 Megabyte Hard Disk Drives For The C64
Quick Brown Box: An 8k Read/Write Cartridge
1540/1541 Drive Alignment System
Astrology Program For Commodore 64
Multiplex Eight RS232 Ports Onto A Single X.25 Line
/SPEEDPAK/ Speedscript Enhancer
Speed-Plus Speedscript Enhancer

TransBASIC Installment #10 18
The Atari ST Notebook 24
BOOT Basics Jim Butterfield on C128 BOOT 28
New Loops: The C128 Stack 30
Eliminating SAVE@ and Other 1541 Bugs 33
FORMAT TRACK 36 36
An Amiga Printer Cable a hardware to hardware project 38
Save SYMASS Symbols a SYMASS Assembler Utility 40
Transcribe 64 a REL file copier utility 42
Animals a simple expert system that's also a game 46
Break Key 64 interrupt without breaking upl 50
MOVE: A Propagating Memory Move Routine 54
Sprite Bit Addressing masking techniques 56
Sprite Numbers display messages the easy way 57
Adding Depth To Your Screens 60
Viewports hi-res and multi-colour mode windows! 61
A C64 Hi-Resolution Draw Routine 66
Hi-Res Search and Print a versatile hires print utility 68
C128 Hi-Resolution Graphics 72
Compu-toons 75

Note: Before entering programs, see "Verifizer" on page 4

The Transactor
The Tech/News Journal For Commodore Computers

Editor in Chief
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Art Director
John Mostacci

Administration & Subscriptions
Ann Richard

Contributing Writers

Ian Adam
Daniel Bingamon
Neil Boyle
Anthony Bryant
Tim Buist
Jim Butterfield
Gary Cobb
Jack Cole
Jeffery Coons
Pierre Corriveau
Robert V. Davis
Elizabeth Deal
Frank E. DiGioia
Yijun Ding
Paul T. Durrant
Michael J. Erskine
Jack Farrah
R. James de Graff
Jim Grubbs
Tom Hall
Bob Hayes
John Jay Hilfiger
Andy Hochheimer
John Holttum
David Hook
Robert Huehn
Tom Hughes
David Jankowski
Chris Johnson
Mark Jordan
Clifton Karnes
Gary Kiziak
Jesse Knight
James E. LaPorte
William Levak
James A. Lisowski
Jack Lothian
Scott Maclean
Steve McCrystal
Stacy McInnis
Jim McLaughlin
Steve Michel
Terry Montgomery
Michael Mossman
Gerald Neufeld
Noel Nyman
Dave Pollack
Richard Perrit
Terry Pridham
Raymond Quirling
Glen Reesor
Gary Royal
John W. Ross
John Russell
Louis F. Sander
Fred Simon
Perry Shultz
P. A. Slaymaker
Edward Smeda
Darren J. Spruyt
Nick Sullivan
Zoltan Szepesi
Karel Vander Lugt
Audrys Vilkas
Andrew Walduck
Steven Walley
Jack Weaver
Charles Whittern
Evan Williams
Chris Wong

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') has a flat top as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print " flush right " – would be shown as – print "[10 spaces]flush right "

Cursor Characters For PET / CBM / VIC / 64

Down – [q]
Up – [Q]
Right – [ ]
Left – [Lft]
RVS – [r]
RVS Off – [R]
Insert – [ ]
Delete – [t]
Clear Scrn – [ ]
Home – [s]
STOP – [c]

Colour Characters For VIC / 64

Black – [P]
White – [e]
Red – [ ]
Cyan – [Cyn]
Purple – [Pur]
Green – [ ]
Blue – [ ]
Yellow – [Yel]
Orange – [A]
Brown – [ ]
Lt. Red – [ ]
Grey 1 – [ ]
Grey 2 – [ ]
Lt. Green – [ ]
Lt. Blue – [Z]
Grey 3 – [Gr3]

Function Keys For VIC / 64

F1 – [E]
F2 – [ ]
F3 – [F]
F4 – [J]
F5 – [G]
F6 – [K]
F7 – [H]
F8 – [ ]

Please Note: The Transactor has a new phone number: (416) 878 8438

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number **6342.** USPS **725-050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209 **ISSN# 0827-2530.**

The Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64) are registered trademarks of Commodore Inc.

Subscriptions:
Canada $15 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 878 8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

Quantity Orders:

U.S.A. Distributor:
Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381
(or your local wholesaler)

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555
(or your local wholesaler)

CompuLit
PO Box 352
Port Coquitlam, BC
V5C 4K6
604 941 7911

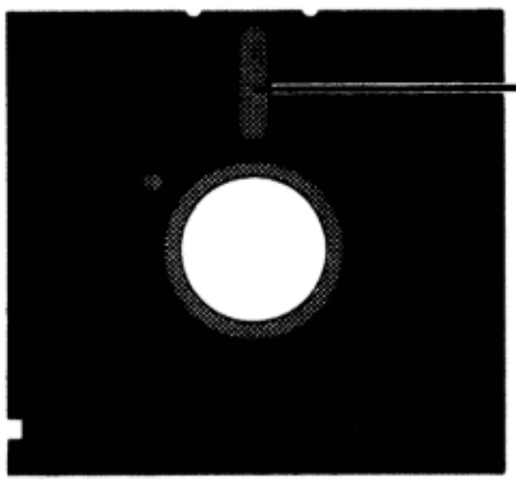
Norland Communications
251 Nipissing Road, Unit 3
Milton, Ontario
L9T 4Z5
416 876 4774

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4 Issues 04, 05, 06, Vol 5 Issues 03, 04
Still Available: Vol. 4: 01, 02, 03. Vol. 5: 01, 02, 04, 05, 06. Vol. 6: 01, 02, 03, 04, 05, 06. Vol. 7: 01, 02

Back Issues: $4.50 each. Order all back issues from Milton HQ.

Editorial contributions are always welcome. Writers are encouraged to prepare material according to themes as shown in Editorial Schedule (see list near the end of this issue). Remuneration is $40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos or illustrations will be included with articles depending on quality. Authors submitting diskettes will receive the Transactor Disk for the issue containing their contribution.

All material accepted becomes the property of The Transactor. All material is copyright by Transactor Publications Inc. Reproduction in any form without permission is in violation of applicable laws. Please re-confirm any permissions granted prior to this notice. Solicited material is accepted on an all rights basis only. Write to the Milton address for a writers package.
The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs. Programs listed in The Transactor are public domain; free to copy, not to sell.



Start Address

The Games Game

TSE – Toronto Stock Exchange? Nope, not this time. Today it stands for "Toronto Software Express" and I hesitate to even mention it. Although what I have to say will not be pretty, I regret giving free publicity to a thorn.

Recently I saw a program demonstrated. It was a game, and a rather professional looking game with high resolution screens that obviously required considerable time and effort. I don't recall the name of the game, but what I remember most was the opening screen. It too must have taken considerable time and effort. It proudly displayed the name of the person who unravelled the protection on the game, the letters "TSE" in large hi-res characters, and a hi-res message that scrolled from right to left containing the names and numbers of TSE bulletin boards in the immediate area. The next screen was a list of no less than 10 more numbers for boards also affiliated with the TSE.

From any of these numbers, one can use their computer to call and download software of all type and make at absolutely no charge. True, most of the programs are public domain, but many are not. I could fill the rest of this page with words I'd like to use describing this sort of activity. Words like deplorable, detestable, despicable, and immoral don't even begin to express how I feel about this.

Since the operators of these services collect no fee, they would probably argue "no wrong doing". However, the manufacturers of these programs are also collecting nothing for these copies. During a year they may spend thousands on development and production, not to mention taxes, overhead, shipping, billing, and a host of other expenses.

All of this has been said before, I know. Likewise, the question arises once again, "would they have bought it if they couldn't get it for free?" Probably not. But what do you suppose would happen if you loaded the latest Whitney Houston tape into a high speed duplicator and started passing the copies out to anyone willing to come and get one. Many people, myself included, probably wouldn't bother. But I suspect someone would ask you to stop, myself included.

Now I'd be willing to bet there isn't one computer hobbyist out there, regardless of the brand name on their equipment, without possession of at least one program currently for sale that they have used without paying for. Getting something for nothing is human nature. But it's also an effective sales tactic which manufacturers have just started using. We all know that without a manual, many programs are virtually unusable. Spreadsheets, wordprocessors, databases, assemblers, and many others are protected this way because sending out free manuals can get expensive. Games seem to be the most vulnerable, especially games which require little or no instruction. Offers like free updates, discounts on other products, exquisite colour manuals, contests for trips or cash prizes, and other more imaginative incentives are planting the idea that "I can go looking for a free one, but if I buy one I also get. . ."

Except it doesn't always work. There are still those who don't care about updates or manuals, or can't get away to Australia. The freebie is just too tempting. So what should be done? Many online services have made themselves responsible for software available on their systems. Their users are warned not to upload copywritten software and those who do can usually be identified. Since these organizations have much to lose (ie. everything), they are less reluctant to invoke disciplinary measures on the culprit than would a basement operation. Action like publicly announcing termination of access privileges is usually enough to eliminate further instances.

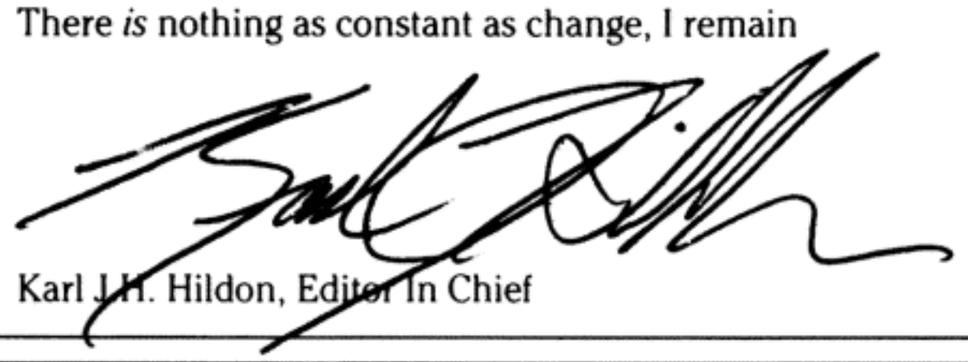
That leaves the bulletin boards, which are so often run by young hobbyists on phone lines billed to their parents. They obtain a BBS program (public domain BBS software is readily available) and load it up with as many programs as their system will handle. Gaining popularity is one of their main objectives. Naturally. But even though there are hundreds of great programs in the public domain, they assume they have nothing to lose by offering proprietary software. Wrongo! New legislation is close to becoming, or may already be law. And if some manufacturer complains, YOU may be first in line to test it out.

If the RCMP came knocking on my door asking for help, would I oblige? You bet I would. But on certain conditions only. First of all, I don't have copies of those phone numbers to offer, but that doesn't mean I couldn't find them. I would need proof of plans for a nationwide campaign. Toronto isn't the only center for this activity. Next, I would need a guarantee that criminal charges would not be strewn about left and right. In my opinion, criminal charges are too easy to get and not easy enough to get rid of. Most of these BBS operators are bright young individuals with brilliant careers ahead of them. Much as I deplore this abuse of freedom, new laws can be easily abused too as law enforcement agencies go looking to "set an example". There are probably other boards run by those old enough to know better, but I would still refuse to participate in ruining the future of talented Canadians without first ensuring them the opportunity to correct their own mistakes.

If you're involved, get out while you can. Keep your BBS, but eliminate the programs that others depend on for income. . . income that keeps them from changing jobs! Besides, you may not be given an early warning. Large corporations often include the cost of penalty before engaging in any covert activity. But take a moment to consider what effect a $15,000 fine might have on your personal savings.

I don't have all the answers, but who knows – there just may be a manufacturer out there willing to investigate the potential for supporting the BBS method of local distribution. For those resourceful enough to make an effort, you may just be able to turn a perilous situation into a profitable one – legally! AND, I'd be only too happy to act as a catalyst, because. . .

There *is* nothing as constant as change, I remain



Karl J.H. Hildon, Editor In Chief

Using "VERIFIZER"

The Transactor's Foolproof Program Entry Method

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two–letter code for each program line which you can check against the corresponding code in the program listing.

There are two versions of VERIFIZER on this page; one is for the VIC or C64, the other for the C128. Enter the applicable program and RUN it. If you get a data or checksum error, re–check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 828 to enable the C64/VIC version (turn it off with SYS 831)
or SYS 3072,1 to enable the C128 version (off with SYS 3072,0)

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

Note: If a report code is missing (or "--") it means we've edited that line at the last minute which changes the report code. However, this will only happen occasionally and usually only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re–check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors (eg. POKE 52381,0 instead of POKE 53281,0), but ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: C64 VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm–start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

VERIFIZER for C64 and VIC–20

```
KE  10 rem* data loader for ' 'verifizer' ' *
JF  15 rem vic/64 version
LI  20 cs=0
BE  30 for i=828 to 958:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
FH  60 if cs<>14755 then print' '***** data error *****' ': end
KP  70 rem sys 828
AF  80 end
IN  100 :
EC  1000 data  76,  74,   3, 165, 251, 141,   2,   3, 165
EP  1010 data 252, 141,   3,   3,  96, 173,   3,   3, 201
OC  1020 data   3, 240,  17, 133, 252, 173,   2,   3, 133
MN  1030 data 251, 169,  99, 141,   2,   3, 169,   3, 141
MG  1040 data   3,   3,  96, 173, 254,   1, 133,  89, 162
DM  1050 data   0, 160,   0, 189,   0,   2, 240,  22, 201
CA  1060 data  32, 240,  15, 133,  91, 200, 152,  41,   3
NG  1070 data 133,  90,  32, 183,   3, 198,  90,  16, 249
OK  1080 data 232, 208, 229,  56,  32, 240, 255, 169,  19
AN  1090 data  32, 210, 255, 169,  18,  32, 210, 255, 165
GH  1100 data  89,  41,  15,  24, 105,  97,  32, 210, 255
JC  1110 data 165,  89,  74,  74,  74,  74,  24, 105,  97
EP  1120 data  32, 210, 255, 169, 146,  32, 210, 255,  24
MH  1130 data  32, 240, 255, 108, 251,   0, 165,  91,  24
BH  1140 data 101,  89, 133,  89,  96
```

C128 VERIFIZER (40 column mode)

```
PK  1000 rem * data loader for ' 'verifizer c128' '
AK  1010 rem * commodore c128 version
JK  1020 rem * use in 40 column mode only!
NH  1030 cs=0
OG  1040 for j=3072 to 3214: read x: poke j,x: ch=ch+x: next
JP  1050 if ch<>17860 then print ' 'checksum error' ': stop
MP  1060 print ' 'sys 3072,1: rem to enable' '
AG  1070 print ' 'sys 3072,0: rem to disable' '
ID  1080 end
GF  1090 data 208,  11, 165, 253, 141,   2,   3, 165
MG  1100 data 254, 141,   3,   3,  96, 173,   3,   3
HE  1110 data 201,  12, 240,  17, 133, 254, 173,   2
LM  1120 data   3, 133, 253, 169,  38, 141,   2,   3
JA  1130 data 169,  12, 141,   3,   3,  96, 165,  22
EI  1140 data 133, 250, 162,   0, 160,   0, 189,   0
KJ  1150 data   2, 201,  48, 144,   7, 201,  58, 176
DH  1160 data   3, 232, 208, 242, 189,   0,   2, 240
JM  1170 data  22, 201,  32, 240,  15, 133, 252, 200
KG  1180 data 152,  41,   3, 133, 251,  32, 135,  12
EF  1190 data 198, 251,  16, 249, 232, 208, 229,  56
CG  1200 data  32, 240, 255, 169,  19,  32, 210, 255
EC  1210 data 169,  18,  32, 210, 255, 165, 250,  41
AC  1220 data  15,  24, 105, 193,  32, 210, 255, 165
JA  1230 data 250,  74,  74,  74,  74,  24, 105, 193
CC  1240 data  32, 210, 255, 169, 146,  32, 210, 255
BO  1250 data  24,  32, 240, 255, 108, 253,   0, 165
PD  1260 data 252,  24, 101, 250, 133, 250,  96
```

Bits and Pieces

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in the Bits & Pieces column, we'll credit you in the column and send you a free one-year's subscription to The Transactor

Head Cleaning For The 2031

D. Tomblin
Parksville, BC

If you own a Commodore disk drive and would like to use a head cleaner disk on it you must run the drive motor for about 45 seconds. Since Commodore didn't see fit to include a "run disk motor for about 45 seconds" command we must look elsewhere. Luckily there is a command that will allow you to execute programs within the drive. Using "m-e" chr$(addr lo) chr$(addr hi) we can turn the drive motor on and off at will. Since Peter Boisvert wrote a program to do the job on a 1541 (Bits and Pieces Trans.V6/3) I will give you the 2031 version. The 1541 won't work with this program. In case you missed Peter's program, substitute chr$(126)chr$(249)in line 50 for use with the 1541.

```
MA  10 print chr$(14); " q This is to be used on a
        2031 type "
OL  15 print "  drive only. Do you wish to continue? "
HJ  20 print: print "              (Y/N) "
NJ  25 poke 198,0: rem  vic and 64
AH  26 rem  use poke 158,0 on 2.0-4.0 pets
BO  30 get a$: if a$ = "" then 30
LF  35 if a$<> " y " then end
GO  40 open 15,8,15
MC  50 print#15, " m-e " chr$(204)chr$(249)
GA  60 print " Sqq      Press Any Key To Stop Motor "
NA  70 get a$: if a$ = "" then 70
IL  80 print#15, " m-e " chr$(54)chr$(250)
BA  90 close 15
EG  100 end
```

SuperGET

Duane Barry, Cambridge Ont.

Here's a very useful subroutine which will use the GET statement instead of an input. The subroutine is inside a tiny demonstration program.

To use the subroutine, first set the variable MX to the maximum number of characters to put into the input field, then GOSUB140. The delete key works so that you can only delete as far as the beginning of the field. When the RETURN key is pressed, the input string is returned in the variable C$.

Note: In lines 150, 210, and 260, "[Logo+p]" is an underline character obtained with the Commodore Logo key and an unshifted P.

```
ID  10 rem* super get statement *
BN  20 rem duane barry - cambridge, ont.
IB  30 print " name: ";
MJ  40 mx = 15     :rem set max input length
OO  50 gosub 140 :rem call super get
KO  60 print " q hello  " c$
GE  70 end
EM  80 :
KM  100 rem** super get subroutine **
LL  110 rem set max input length in 'mx'
DF  120 rem input string returned in 'c$'
GP  130 :
AI  140 l = 0: a$ = " " : c$ = " "
IJ  150 print " [Logo + p][Crsr Left] " ; :rem print
        fake cursor
NP  160 get a$: if a$ = "" then 160
DL  170 if a$ = chr$(20) and l = 0 then 160
OF  180 if a$ = " [crsr left] " or a$ = " Q " or a$ = " q "
        or a$ = " | " or a$ = chr$(19) then 150
DL  190 if a$<>chr$(20) and a$<>chr$(148) then 220
BK  200 rem delete last char
AG  210 l = l-1: print " [spc, 2 crsr lefts][Logo + p]
        [crsr left] " ; : c$ = left$(c$,l): goto 160
LO  220 if a$ = chr$(13) then print "  " :return
KI  230 if l = mx then poke198,0: goto160
AJ  240 print a$;          :rem print new char.
BD  250 poke 212,0         :rem turn off quote mode
LF  260 print " [Logo + p][crsr left] " ; :rem print cursor
BG  270 c$ = c$ + a$       :rem add char to string
BJ  280 l = l + 1          :rem add to char count
FM  290 goto 160           :rem get next char
```

Turning off REMOTE 64

John Obeda
London, Ont.

John Obeda called to point out a problem with REMOTE 64 (Volume 6 Issue 2): it doesn't quite work as advertised if you try turning it off and re-initializing it. The method given to disable REMOTE 64 was to re-set the IRQ vector. Actually, you'll have to restore a whole bunch of vectors to ensure safe re-initializing. Fortunately, there is a Kernel routine which will re-set all system vectors to their default state. The routine is in the jump table at $FF8A (65418 decimal). To turn off REMOTE 64, just:

```
SYS 65418
```

Then SYS 49152 to turn REMOTE 64 back on when you need it.

Freakout and Farzoid

Marc Moorcroft
Toronto, Ont.

Here's a couple of just-for-fun curiosities for your 64. The first one, freakout, restores normal control of your machine, but adds the dubious benefit of strange sound effects to accompany your every command. "Farzoid" is a graphics toy which switches between background colours quickly enough to out-speed the raster beam and produce a dazzling display.

Freakout:

This C-64 program sets up an interrupt routine that gets the saved processor registers from the stack and puts them into the voices of the SID chip.

```
OM  90 rem freakout by marc moorcroft
DN  100 s=54272: l=49152: poke s+24,15
HD  110 for t=s+4 to s+18 step7:poke t+1,0:
        poke t+2,240:poke t,17:next
KO  120 for t=l to l+21:read a:poke t,a:next
AP  130 poke 56334, peek(56334) and 254
LC  140 poke 789,l/256:poke 788,l-peek(789)*256
EJ  150 poke 56334,peek(56334)or1
JJ  160 data 186, 189, 1, 1, 141, 1, 212, 189, 2, 1
HF  170 data 141, 8, 212, 189, 3, 1, 141, 15, 212, 76,
        49, 234
```

Try listing a long program while holding down the CTRL key. For more interesting sounds, try increasing the rate of the interrupts:

```
poke 56325,1
```

or

```
poke 56324,30: poke 56325,0
```

Farzoid:

Pick two colours (numbers 0 through 15) for the first prompt, then a speed value for the next. The values 0,5,14,19,25,38,49 and 50 give good results. To stop the program and try a new speed value, hit RESTORE. Press RETURN at the speed prompt to go back to the colour prompt, and RETURN at the colour prompt to exit the program.

```
OB  100 for t=828 to 868: reada: poke t, a: next
NE  110 print" qr farzod R by marc moorcroft qq "
NB  120 a=-1:input" QQ flip between which
        colours[7 spcs, 7 crsr lefts] ";a,b
BO  130 ifa<0orb<0ora>15orb>15then end
EL  140 poke 863,a: poke868,a and not b or b and
        not a: print
MP  150 d=-1: input" Q delay (0-121)[7 spcs,
        7 crsr lefts] ";d
HF  160 ifd<0 or d>121 then 120
HE  170 for t=869 to 868+d: poke t,234: next
EF  180 poke 869+d,16: poke 870+d,249-d:
        a=peek(53281)
CG  190 sys 851: poke 53281,a: goto 150
FO  200 data 173, 13, 221, 169, 4, 141, 13, 221, 169,
        71, 141, 24, 3, 169, 254, 141, 25, 3
HO  210 data 88, 104, 104, 104, 96, 120, 169, 60, 141,
        24, 3, 169, 3, 141, 25, 3
GA  220 data 169, 3, 141, 33, 208, 73, 1
```

Device Not Present Update

John Menke
Mt. Vernon, IL

Dave Pollack's cure for the 'DEVICE NOT PRESENT' bug in the C-64 operating system is good news (Volume 6, Issue 6, page 6). It works well in my tests, after a slight modification based on my observation that the value of ST is not always exactly -128 when a serial bus device is disconnected. I have found values of -125 and -128, and presumably other values are possible. Probably any negative value of ST correctly signals that a serial device is not present.

"Display T&S" Fix

John Houghton
Collingwood, Ont.

I use the program DISPLAY T&S on this disk that came with my 1541 quite a bit. Recently I had some weird problems when accessing some files I was studying. I would display the directory track to find the first track and sector, and when the program displayed the next track and sector for the block, it was either a wrong or illegal track or sector. I could only see the first track and sector then CRASH.

What was frustrating was that the files would load fine so DOS was seeing the right numbers, and a DISK DOCTOR program I

have read out the correct information all the time. Obviously something was wrong with the program DISPLAY T&S.

The fix is in line 270. In line 410, the program memory-reads $0500 (buffer #2), but the OPEN command in line 270 only requests ANY buffer for use. Change line 270 to read:

```
270 open 2,8,2,"#2" : gosub 650
```

This will reserve buffer two for your use. For some reason the program was getting the wrong track and sector info from the buffers.

A note: this is only happening to me on one disk, which leads me to think that re-NEWing the disk has done something, because I did re-NEW the disk before its use. If this is happening to anyone else the fix is above. Works for me, so far, every time.

Underline Cursor for the 64

Ben Russel
North Sydney, NS

Here's a program to give you an underline-type cursor on the 64 whenever the character under the cursor is a space. The program also sets the background, border, and character colours after a RESTORE.

Following are some POKEs for using the cursor program.

For users of the Fastload cartridge:

```
poke 49189, 106: poke 49190, 223
```

To change background and border colours:

```
poke 49176, colour
```

To change the cursor colour:

```
poke 49184, colour
```

To change the blink speed of the cursor (default is 20; lower numbers = faster):

```
poke 49209, n
```

```
ON 10 rem* data loader for "cursor" *
LI 20 cs=0
KG 30 for i=49152 to 49257:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
KA 60 if cs<>13386 then print "!data error!" : end
DD 70 sys 49152
AF 80 end
```

```
IN 100 :
CB 1000 data 169, 11, 141, 2, 3, 169, 192, 141
CK 1010 data 3, 3, 96, 120, 169, 39, 141, 20
IC 1020 data 3, 169, 192, 141, 21, 3, 88, 169
OO 1030 data 0, 141, 32, 208, 141, 33, 208, 169
KE 1040 data 9, 141, 134, 2, 76, 131, 164, 32
HI 1050 data 234, 255, 76, 48, 192, 76, 97, 234
ME 1060 data 165, 204, 208, 249, 198, 205, 208, 245
FG 1070 data 169, 20, 133, 205, 164, 211, 70, 207
LL 1080 data 174, 135, 2, 177, 209, 176, 17, 230
LE 1090 data 207, 133, 206, 32, 36, 234, 177, 243
ME 1100 data 141, 135, 2, 174, 134, 2, 165, 206
EB 1110 data 201, 32, 240, 9, 201, 100, 240, 5
IP 1120 data 73, 128, 76, 94, 234, 73, 68, 76
MP 1130 data 98, 192
```

Complete Joystick Keyboard Equivalents

Robert Blain
NSW. Australia

In a previous Bits and Pieces, we gave ways to simulate some joystick positions via the keyboard. From Robert Blain comes this complete list. The left column tells which key or keys to press to give the joystick equivalent in the right column.

JOY1

CTRL	= West
←	= South
1	= North
2	= East
1 + 2	= North East
← + 2	= South East
← + CTRL	= South West
1 + CTRL	= North West
Space	= Fire

JOY2

Space + C	= West
Space + Z	= South
Space + F1	= North
Space + B	= East
Space + C + Z	= North East
Space + B + Z	= South East
Space + B + F1	= South West
Space + C + F1	= North West
Space + M	= Fire

Define Extra C-128 Keys

Richard Young
Greenwood, NS

The Commodore 128 System Guide covers function key definitions, using the KEY command, but the HELP and RUN (SHIFT RUN/STOP) keys can also be defined. All of these definitions

are located from $1000 to $10FF (4096–4351 decimal). It is easiest to re-define HELP and RUN before re-defining the function keys because the definitions move in this area of memory according to the length of the definitions.

For example, to re-define RUN from DLOAD "* and RUN to simply "RUN:", memory locations $103F to $1047 (4159–4167) must be altered:

Enter the MONITOR and display **M 103F 1047**
You will see the dL"* and RUN in ASCII.
Change $103f to $1047 to read:

00 00 00 00 52 55 4E 3A 0D

exit from the monitor (X) and define other keys if desired. SAVE your key definitions from the monitor with:

S "KEYS",8,1000,1100

or from BASIC with

BSAVE "KEYS",B0,P4096 TO P4351

The key definitions can now be recovered at any time with:

BLOAD "KEYS"

Setting the B–128 Interrupt Rate

Elizabeth Deal
Malvern PA

Interrupts on the B–machines are handled by the tri–port chip at $de00. It appears that any valid cia2 source, in addition to the TOD–alarm, can be used to interrupt.

Normally, IRQs occur at the power line frequency (bit 0 at $de02). They can be turned off and replaced by another source, for instance, timer–A falling through zero, as on the c64. It's a pest, but can be done.

You can type the bytes under the ******** using the monitor. To get it going, use:

bank 15: sys 7*256

The little reset button kills the project.

One use might be U.K. Software which relies on the 50/sec irqs. You may try it with Superscript (which bounces keys due to the 60 HZ irq): start the code using the SYS call above (not monitor!), then load and run Superscript as you normally do. My keyboard does not bounce the keys anymore and always types correct characters.

```
        ********
f0700 78          sei
f0701 ad 07 de    lda  $de07  ;clear latch & air
f0704 8d 07 de    sta  $de07
f0707 a9 40       lda  #$40   ;set timer a for 50hz ($9c40)
f0709 8d 04 dc    sta  $dc04  ;(60hz = $8235)
f070c a9 9c       lda  #$9c
f070e 8d 05 dc    sta  $dc05
f0711 a9 81       lda  #$81   ;icr–enable ta irqs only
f0713 8d 0d dc    sta  $dc0d
f0716 ad 0e dc    lda  $dc0e  ;cra–leave 50/60hz flag alone for tod
f0719 29 80       and #$80    ;start ta in continuous mode
f071b 09 01       ora  #$01
f071d 8d 0e dc    sta  $dc0e
f0720 ad 05 de    lda  $de05  ;tri–port irq mask . . .
f0723 29 fe       and #$fe    ; ignore power line flips
f0725 09 40       ora  #$40   ; enable cia2 irqs instead
f0727 8d 05 de    sta  $de05
f072a a9 36       lda  #$36   ;switch irq routine to (*)
f072c 8d 00 03    sta  $0300
f072f  a9 07      lda  #$07
f0731 8d 01 03    sta  $0301
f0734 58          cli
f0735 60          rts
f0736 a5 01 (*)   lda  $01    ;irq patch
f0738 48          pha
f0739 ae 07 de    ldx  $de07  ;look at irq flag
f073c 8a          txa
f073d 29 04       and #$04    ;was it cia2"?
f073f  d0 04      bne $0745   ;yes, go around
f0741 8a          txa         ; no. . .
f0742 4c f5  fb   jmp $fbf5   ; . . .do usual stuff
f0745 ea          nop         ;yes – try 'inc $d043' here
f0746 ea          nop
f0747 ea          nop
f0748 ad 0d dc    lda  $dc0d  ;get/clear cia2–irq flag
f074b 29 02       and #$02    ;is ieee–timeout bit set"?
f074d f0  10      beq $075f   ; nope, go around
f074f  a9 00      lda  #$00   ; yup, force it on
f0751 8d 06 dc    sta  $dc06  ; so that disk routines can see
f0754 8d 07 dc    sta  $dc07  ; this bit.
f0457 ad 0f  dc   lda  $dc0f  ; set timer–b for one shot mode
f045a 09 18       ora  #$18   ; so it won't time out all the time
f045c 8d 0f  dc   sta  $dc0f  ; but disk will still see it once
f045f  4c 81 fc   jmp $fc81   ;do keyboard scan, etc. . . rti
```

More B–128 Info

Some additions to the Transactor V4 #5 Issue for Protecto/CBM B–128 version

$de02: irq latch. poke zero to clear 1 bit, read $de07 to clr all

7	6	8	4	3	2	1	0
not	not	irq	acia	cia1	cia2	ieee	60hz
used	used	pend		ip		srq	pwr

$de06: cb is unconnected (both bits are high)

$dc0d: bit 1, timer b (cont.mode) is set for ieee 65ms timeouts. bit 1 is tested only, no irqs.

$dc0e: cia2 cra. bit 7 is zero (1 in u.k.) tod will use 60hz. bit 3 is set at ioinit time – ta one shot and never used.

B–Series Sound

Edward Shockley
Deerfield Beach, FL

Those of you who own a B–series machine may not realize that it contains the same 6581 SID chip as the C64, C128, and SX64! And you can take a BASIC 2.0 program written for the SID and run it on the B–series with little modification. Simply add 1536 ($600 hex) to the C64 program's SID memory locations. Leave the values alone, they're the same on all machines.

For instance, the SID starts at 54272 ($D400) on the C64 and at 55808 ($DA00) on the B–series. Adding 1536 to 54272 yields 55808, the B–series location. Follow through the program, adding 1536 to each POKE of the SID chip. If the program assigns the voices separate from playing them, you may need to add a BANK 15 statement just before both routines.

Of course, you can convert the B–series programs to the C64 by reversing the process.

Find A Program's Start and End Address

Russ Thomas
Bridgewater, NS

The following program will give the start and end address of a program on disk. The program finds the end address in a rather unique way. Instead of walking through a program to the end, this program attempts to verify the file. Because the program is not in memory, a verify error occurs (the program being checked MUST NOT BE IN RAM for this to work). When the program ends due to the error, it continues due to some commands poked into the keyboard buffer. At this point, locations 174 and 175 hold the end address, which is converted to hex and printed out.

Note: the cursor movements in lines 130 and 140 must be typed in exactly as listed, or the program will not continue after it hits the verify error.

```
II  20 print " q this program will give the start "
GI  30 print " and end address of a program "
FB  50 input " q what is the filename " ;f$
EG  60 open 2,8,2,f$ + " ,p,r "
JK  70 get#2,lo$: lo$ = lo$ + chr$(0)
FG  80 get#2,hi$: hi$ = hi$ + chr$(0)
CK  90 close 2 :la = asc(lo$) + 256*asc(hi$)
```

```
OO  100 print " Sqqqq]]]]]]]]]]]]]]qqqqqqq working "
DA  110 poke198,2:poke631,13:poke632,13:
DC  115 oc = peek(646): rem save old chr colr
KB  120 print " s " :poke646,peek(53281)and15
EM  130 print " qqqqqqqqq goto160 " :print " s "
JN  140 print " qqq verify " ;chr$(34);f$;chr$(34);
        " ,8,1 QQQQQ "
FC  150 poke198,3:poke631,13:poke632,13:
        poke633,13:end
EE  160 ea = peek(174) + 256*peek(175)-1
CP  170 poke 646,oc: rem restore char color
OO  180 print " ]qq]]qr " f$:print " R " :print " q "
JP  190 h$ = " 0123456789abcdef "
PP  200 x = la
DD  210 for i = 3 to 0 step-1
PD  220 n% = x/(16↑i):x = x-n%*16↑i
DM  230 he$ = he$ + mid$(h$,n% + 1,1)
EP  240 next
FE  250 p$ = " 0123456789abcdef "
DC  260 z = ea
PG  270 for i = 3 to 0 step-1
JI  280 n% = z/(16↑i):z = z-n%*16↑i
PD  290 pe$ = pe$ + mid$(p$,n% + 1,1)
AD  300 next
ID  310 print "  start address : " la; " dec.  " ;he$; " hex "
EC  320 print "  end address   : " ea; " dec.  " ;pe$; "  hex "
CI  330 clr
MI  340 print " qq another file? y/n "
LL  350 get a$: if a$ = "" then 350
LP  360 if a$ = " y " then 50
CH  370 end
```

Finding Relative File Record Lengths

Use this short program to find the record length of a relative file on a 1541 or 2031 drive:

```
10 input " file name " ;f$
20 open 15,8,15
30 open 1,8,9,f$
40 print#15, " m-r " chr$(200)chr$(0)chr$(1)
50 get#15,rl$
60 print " record length of file " ;f$; "  is " ;asc(rl$ + chr$(0))
70 close 7: close 15
```

For 8050, 4040, or 8250 drives, change the (200) in line 40 to (114).

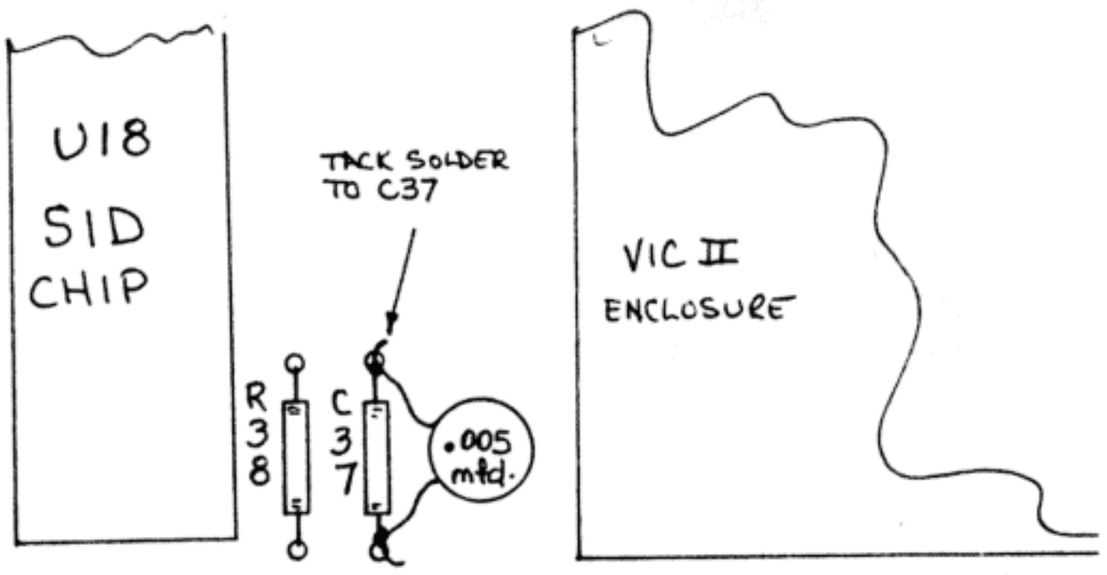
SID Whistle–Killer

Peter Vdovich, Pickering, Ont.

Recently I purchased a Voice Messenger speech synthesizer to use with my 64, and was rather dismayed by the high–pitched whistle that was present when it set the level of the SID chip. (This whistle is also present, to a lesser degree, any time the

chip provides maximum output.) A few tests confirmed that this racket does indeed come directly from the SID.

Because I work as a repair technician (and I'm a cheapskate), I decided to try and eliminate the whistle myself. I opened up the computer and installed a .005 mfd. capacitor in parallel with C37 on the circuit board. (The added capacitance attenuates high frequencies. Different capacitor values will produce different results, but .005 mfd seems best.) The operation was fast and easy, and although the whistle is not completely gone it is greatly reduced. The synthesizer sounds much better too!



Disk Rescue **H.C. Doennecke, Tulsa, OK**

When used with certain disks, My 1541 became increasingly confused when loading or saving. The red light would blink and the end stop would rap repeatedly, after which the operation might or might not be successful. Whenever I checked the error message it would be Number 21, "No sync character". However, my purchased software loaded normally and various checks said the drive was OK. I decided the disk surfaces were going bad and discarded a couple of disks before I awoke to the fact that the disks were dragging in their envelopes. Now whenever a disk starts to do this I bend open one end of the envelope, carefully remove the disk, and root around in the envelope, especially to the edges, with a plastic drafting triangle. Then I replace the disk, add a bit of tape to hold the end flap down, and the disk works as good as new.

All of my disks that have had this problem have been Memteks which have been used about a year and a half.

LOAD By Track and Sector **S.L. Mickelson, London, England**

It is known that using LOAD"*",8 with the 1541 will load the previously loaded program. The drive keeps pointers in RAM which point to the first track and sector of the last file loaded. So when LOAD"*",8 occurs, the drive doesn't access the directory first but instead goes to the track and sector stored in these RAM pointers and begins the load from there. By changing the pointers, we can get the drive to load any program we wish, making all sorts of things possible.

For instance, you could write a menu program which would load programs from disk without accessing the directory; the starting track and sector of all programs on disk could be known by the menu program, and the programs don't even need to be in the disk directory. This would speed up program loads, and make for an un-cluttered directory as well.

To load a program starting at a given track and sector (using LOAD"*",8), use the program below:

```
10 t=17: s=0: rem - example first track and sector of file to
   be loaded
20 open 15,8,15,"i0"
30 print#15,"m-w"chr$(126)chr$(0)chr$(1)chr$(t)
40 print#15,"m-w"chr$(111)chr$(2)chr$(1)chr$(s)
50 load"*",8
```

To implement the menu program mentioned above, you'll have to start with a normal disk and find out the start track and sector of each program. There are a few public domain programs which will give you the track/sector information, or you can use this short program:

```
1 rem  displays the name + first track
2 rem  and sector of every file on disk
10 open 15,8,15,"i0": close 15
20 open 1,8,3,"$0"
30 for i=1 to 254: get#1,a$: next
40 f$="": m=m+1: get#1,a$,t$,s$: t$=t$+chr$(0):
   s$=s$+chr$(0)
50 for i=1 to 16: get#1,a$: f$=f$+a$: next
60 for i=1 to 10: get#1,a$: next: if st<>0 then close 1: end
70 if t$<>chr$(0) then print f$"  t="asc(t$)"  s="asc(s$)
80 get#1,a$: if m<8 then get#1,a$,a$: goto 40
90 m=0: goto 40
```

A Use For The Periodic Table! **Robert G. Tischer, Starkville, Mississippi**

The Complete Commodore Inner Space Anthology finds itself in daily use in our shop and does so because of its many thoughtfully constructed and easily usable tabulations.

One extra use I've found is in providing interesting disk ID's.

For this purpose, the periodic table on page 121 is IDeal. It provides over a hundred ID letter combinations, guaranteed to be different, meaningful, useful as a learning aid, and, no doubt, in other yet unrecognized ways.

Moreover, the table itself represents a checking device, useful in determining which ID's have already been used.

In the event that you are a really heavy disk user, the ID list could be expanded easily, still retaining the basic structure of

the periodic table by using the first letter of each element, accompanied by the letters of the alphabet, in order.

This would provide more than 2600 ID's. If you need more than that, you're on your own.

Pythagorean Triplet Generator

R.A. Israelson
Kingston, Ont.

The following program will generate integer solutions to the right triangle according to the theorum of Pythagoras, based on the variables U and V as shown.

Run the program to see if you can determine the required relationship between U and V, if the generated data is to be non–redundant. You can vary the number of solutions produced by changing the value of N in line 130.

```
AB  100 rem pythagorean triplets by rai -86
HJ  110 print" Sqq  u  v   u↑2-v↑2   2uv   u↑2+v↑2"
IJ  120 print" [ 37 '' = '' signs ] "
LK  130 n=10: rem number of solutions
KB  140 dim a(n-1),b(n-1),c(n-1)
PI  150 quit=n: j=0: u=2: v=1
ML  160 u2=u*u: v2=v*v
JM  170 a(j)=u2-v2: b(j)=2*u*v: c(j)=u2+v2
GO  180 print tab(2);u;tab(6);v;tab(15);a(j);tab(25);
        b(j);tab(34);c(j)
BF  190 j=j+1: if j=quit then end
FL  200 if u-v = 1 then goto 220
JP  210 v=v+2: goto 160
GB  220 u=u+1: v=1
HJ  230 if (u and 1)=1 then v=2
NJ  240 goto 160
```

Easy DATA Viewing

Philip Sharman
Calgary, Alberta

Just typed in a program with a lot of DATA statements that now need proofreading? Add this line to the beginning of your program:

```
1 read x: print x: wait 653,1: wait 653,1,1: goto 1
```

Then just hit the SHIFT key and the numbers in the DATA statements will be printed, one by one, on the screen. Much easier than staring at a screen full of numbers! It's even easier than proofreading from a printout.

It'll stop harmlessly with an "out of data" message when it runs out of numbers. Just erase line 1 when you're finished.

Textcraft Files On Amiga's External Drive

Ewan Grantham
Atlanta, GA

According to the Textcraft manual, you can store documents on an external drive, but they must be on the internal drive to open them. Actually, you can establish a pathway to the external drive to both save and open documents.

To do this, put your document disk in the external drive, and the Textcraft disk in the internal drive. Open Textcraft from the external drive by double–clicking the mouse on one of the document icons from the external drive. This will load Textcraft and the selected document. Interestingly enough, this establishes the save pathway to the external drive. To open other documents from the external drive, save the first document to the external drive using:

"df1:<filename>".

After doing this, you can open and save documents on your external drive, reducing the chances of hurting the Textcraft master disk.

C–64 Easy Filename Retrieval

Russ Thomas
Bridgewater, N.S.

In the May, 1986, Volume 6, Issue 06, Jeffrey Coons gives an on–screen one–liner to find the filename after a FILE NOT FOUND ERROR. Try this instead:

SYS 62913

A nice trick – fast, easy to remember, and it prints to the screen. Good for the C64; VIC owners can use SYS 63065.

C128 CP/M ASCII Output

Richard D. Young
Greenwood, N.S.

The additional CP/M documentation and utility disks offered for the Commodore 128 and available from Commodore Business Machines for $29.95 may be worth it for anyone who is going to use CP/M at all. One of the disks contains complete source files and instructions for regenerating the CPM+ system. These files, in source assembler (.ASM), are reasonably well documented with comments and can be an excellent aid to learning the new machine language and the operating system.

After using CP/M for some time, including output to a printer via an interface that converted CBM ASCII to true ASCII, I experienced some frustration with this process of conversion. As a general rule, I have always preferred to convert to true ASCII in software prior to output to the printer. CP/M on the Commodore 128 appeared to offer no alternative until I had a look at the source code.

Among the files is one called CXPRINTE.ASM. This is the printer driver, and contains the routine for converting characters for output to an assumed Commodore printer. It turns out that characters are carried in true ASCII until this conversion routine is called. Specifically, the line in the source code is " CALL CONVERT". Deleting this line, or replacing it with a NOP and a comment, then regenerating the CPM+ system by following the instructions with the source disk was sufficient to allow output of true ASCII to the printer.

Un–Assembler Speed–Up — **David Shiloh, Eugene, OR**

I thoroughly enjoyed Mr. Lothian's Un–Assembler (Jan '86), which does not suffer at all from garbage collection. It is, however, doing extra floating point operations in pass two as it builds the label arrays. The replacement lines below make pass two over five times as fast (for 6K, 14 minutes instead of 81). Mr. Menke (Letters, May '86) should now be able to unassemble 21K of code in just over two hours.

```
530 gosub 2380: p=s-1: gosub 2260: l1(0)=s:
    l1(1)=e+3: lb=1: l2(0)=0
540 rem  print p" Q " to watch pass two
680 for i=lb to 0 step-1: if ad<l1(i) then 710
690 if ad=l1(i) then i=0: goto 710
700 for q=lb+1 to i+2 step-1: l1(q)=l1(q-1):next:
    l1(q)=ad:i=0:lb=lb+1
710 next: return
720 [DELETE]
730 [DELETE]
770 [DELETE]
780 lf=1: for i=0 to lo: if ad>l2(i)then 810
790 if ad=l2(i)then lf=0: i=lo: goto 810
800 for q=lo+1 to i+1: l2(q)=l2(q-1): next:
    l2(q)=ad:lf=0:i=lo:lo=lo+1
810 next: if lf then l2(i)=ad: l0=l0+1
820 return
830 [DELETE]
```

and. . . Line 1430 needs to end with ",2040" for the bit to .byte option

Bigger Stacks For Healthy Assembly

When assembling a large file on the Amiga, the system can crash because of lack of stack space. Before assembling, give yourself more space with the STACK command from CLI. A value of 20,000 should be safe. When in doubt, use more. The command looks like this:

STACK 20000

You may want to put this in the batch file you use for assembling, or even in your Startup–Sequence. You also need a larger stack when running ABasiC from CLI (8000 is safe).

Amiga ABasiC Program Boot — **James Cooper Jr., Fayetville, NC**

When ABasiC (the early Metacompco BASIC) is loaded, it looks for a file named "init.bas", first in the current directory, then in the ":s" subdirectory. If found, ABasiC automatically loads and runs this file. I use this feature to set my screen to 80 columns before I start typing.

As an added extra, ANY script file may be placed in the ":s" subdirectory, as execute looks there if it cannot find your file in the current directory. This way, you do not have to worry if you have changed the current directory when you try to execute the file.

Easy Speed–Up For The 1541 — **Ralph Doncaster, East Bay, N.S.**

For people who are trying to squeeze every bit of speed out of their 1541 or who are just interested in the workings of the disk drive, then this is just for you. Type in, save, and run the following:

```
10 open 15,8,15,"m-w"chr$(94)chr$(0)chr$(2)chr$(1)chr$(41)
20 print#15,"m-w"chr$(100)chr$(0)chr$(1)chr$(3): close 15
```

This speeds up the movement of the read/write head by changing the acceleration and the amount of time between steps. Time to move the head is reduced by more than half, giving speedier program and file access.

Amiga Date-Prompt Startup Sequence — **Benjamin Dobkin, Rego Park, NY**

I use this file as my S/STARTUP—SEQUENCE:

```
echo "    "
echo "        Welcome Master Benjamin!!"
echo "    "
echo "Our last session was on:"
date
echo "    "
date > "con:0/100/350/25/DD-MMM-YY HH:MM:SS [en-
  ter new date:]" >* ?
date
loadwb
endcli > nil:echo "    "
```

This makes a new window containing a date-time prompt (a la IBM), and right after you respond to the prompt it will display the full date and time.

At this point if you press control-d, the startup sequence is broken and you are returned to CLI without going into Work-Bench.

Letters

The Mutant Vic: In a city not so far away, in a time but four years ago, a disenchanted university (biology) grad unable to find work in his own field decided to escape his day time job and purchased his night time dream machine, a 3.5k Vic 20 (for roughly 10x what you would pay for one now).

He subscribed to all the beginner magazines and really enjoyed himself for about six months and then some virus bit him. The Vic grew out of control. It grew to 32k, it learned how to transfer ROM cartridges to tape and grew chip switches. It took over a room in the house. It became so much fun that he started taking a data processing certificate at a community college. Two years passed and the Vic spawned a mutant, a C–64, more powerful than itself. It demanded a disk drive, then a printer, then another disk drive, more languages. And then all of a sudden the biology grad was running a main frame computer with multi partitions for the same company he used to pick and pack orders for. Another year passed and now he was a programmer. Who would have ever guessed that a 3.5k Vic 20 would lead to this. This summer he would graduate from college.

But along the way something else happened. All those magazine subscriptions just weren't pulling their weight anymore. They were fine to start with but they didn't grow, they remained superficial barely scratching beneath the surface. One by one they weren't resubscribed to until only one remained. The Transactor. Please find enclosed my renewal to my subscription of The Transactor and keep up the excellent work.

S.J. Balduck, London, Ontario

In another city not so far away from yours, in a time but four weeks ago, a letter would arrive at the office of a computer magazine. The letter would describe the experiences of a determined young university grad who would become interested in a completely different field than the one printed on his diploma. The letter would turn out to be written by one and the same. A wonderful letter as it would allow the editor of the computer magazine to respond in a writing style he would enjoy immensely.

Thank you for the compliments and best of luck in your new found career. However, (and this probably need not be said) don't be dissappointed about the time you spent learning biology. I too spent a long time obtaining my diploma in electronics engineering. Although I have been lucky to be able to apply SOME of what I learned, I have found that the "paper chase" was the most valuable lesson of all. Learning how to learn or "learning discipline" is a gift that most grads don't realize they've received until the time comes to put it to use. For those still in schools of higher ed, perhaps I've revealed too much. But let me just say this: STAY THERE! What you're seeing now is much less than what you'll get out of it later, I assure you! – K.J.H

Postal Service Blues: What happened to the days of the valued subscriber? You remember; subscribe at a small reduction of cost and get your issue a week before the news stand.

I recently received my notice of renewal for my Transactor magazine. My much awaited Transactor magazine I might add. I have to wait too long to receive them! Tell me how the editors of magazines expect the general buying public to put out a years subscription price and then wait anxiously. Wait for one or two weeks after the magazine has already arrived on the news stand. I've dropped my other subscriptions for the same reason! There really was a time when the subscriber was privileged; receiving his or her copy in advance of the general public.

I still look forward to getting my Transactor. No ads. Lots of substance. So much so, I won't wait a minute longer than I have to! That's the best compliment anyone could receive. . .

James R. Clefstad, Mackenzie, British Columbia

Yours is typical of many letters we receive as well as phone calls and personal comments. We would like to apologize for the delivery problems that have plagued any and all of our readers, but would also like to take this opportunity to describe the process.

When an 80 page Transactor goes on the press, the pages come off in 2 sheets of 32 (16 pages each side), one sheet of 16, plus the cover and any other inserts (eg. the subscription card). All this gets moved over to another rather large mechanical contraption called a "binder".

However, before MacLean-Hunter (our printer) will fire up the binder, they insist we submit our mailing list of subscribers. The list is sorted by postal/zip code and printed on a dot matrix printer (at our office) on regular 132 column paper such that there are four labels across by eleven down on each page. The labels must be very carefully positioned on the page according to specs provided by MacLean-Hunter. The entire stack of subscriber labels are loaded into a hopper at the end of the binder, and the process begins.

The binder machine takes the various sheets and folds them this way and that until each 2-page set is seperated by a fold (a 2-page set is page X plus the page it's attached to at the opposite end of the magazine). At about the half way mark, the folded pieces are stacked in order, and the entire stack is stapled together. The staples actually start out as a spool of wire that is cut and bent into what you see as staples. Next the bound stack is bent in half along the staples, and a "trimmer" comes smashing down and sheers off the folds at the top, bottom, and right hand edge. As you can imagine, the trimming must be done last. If it were done any earlier, each page would be a slightly different length leaving a rather untidy appearance to the three exposed edges of the paper.

About three feet later, the finished magazine enters the labeller. The 4 by 11 sheets of labels are sliced and diced, dipped in glue, and slapped onto the cover. As I metioned earlier, the labels must be carefully positioned on the paper else the slicer/dicer would trim off part of the data. The labelled mags go from there into a bundler. The bundler operator watches the destination of the magazines very carefully so that those going to one province or state don't get bundled with those going to the next province/ state. The bundles (about 20 in each) go onto skids, one skid per province. The U.S. is seperated into 8 "zones", hence 8 skids for delivery to the Buffalo main postal dispatch.

How does this affect delivery? Well, it doesn't. . . we just thought you might like to know. However, it is easier for MacLean-Hunter to do the subscriber copies first. To do them last would mean anticipating the point at which about 10,000 unbound copies remain before switching on the labeller. By doing them first means simply using up all the labels. When the labels are all gone, the rest of the magazines go to the news stand distributor. But remember, we print about 64,000 copies, approximately 10,000 of which go to subscribers. When the last label is pasted on, there are still 54,000 mags to go. This takes several hours. When the last magazine is bound and trimmed, the subscriber copies are usually well on their way (MH doesn't fool around - they don't like having 15 to 20 skids of mail taking up precious floor space). In fact, Canada Post makes regular pick ups at MH en route to their major distribution points. The only mail that goes to the Toronto center is the mail for the Toronto area. The rest goes directly to Montreal, Edmonton, Vancouver, etc. The U.S. mail goes to Buffalo, and from there on to the other zones.

Admittedly, Transactor mail doesn't fill an entire truck load so MH combines it with other titles going to the same places. However, the process is about as streamlined as it can get, and you can bet the entire lot goes out on pretty much the same day. Your complaint, though, is not unjustified - quite the contrary. Except to get subscriber copies out earlier would probably mean warehousing the news stand copies for two weeks to give the others a head start. As you can see, it's one bind after another (pun intended).

A Comparison Of Four Word Processors Revisited: I would like to make a few comments about your article entitled 'A Comparison Of Four Word Processors'.

First of all, in your article it was stated that SpeedScript could only display 40 columns. This is partially true. Through the use of Preview–80, it is possible to scroll around your SpeedScript document using the arrow keys. This is not true 80 columns, but then again it is handy when you want to see what your document will look like before you print it out. Special printer functions such as the underline and bold text modes cannot be displayed. Functions such as headers and footers can be displayed. Preview–80 can be found in the November 1985 issue of Compute!s Gazette (issue 29, vol. 3, no.1).

It was also stated that SpeedScript does not have a spelling checker. Again, this is partially true. In the December 1985 issue of Compute!s Gazette (issue 30, vol. 3, no. 12) a program called SpeedCheck was published. This is a spelling checker, but there is no pre–made dictionary disk. The program allows you to make your own custom dictionary by going through your SpeedScript document, checking for all words that are more than 5 characters in length, and asking if you want this word in your dictionary. In future documents, if the word it is checking is already in the dictionary, it passes it by and goes on to the next word. There are a few understandable shortcomings to this spell checker. For example, what if you exchange 'their' for 'there'.

In a recent Compute! (I'm not positive of the month, but it was the issue with SpeedCalc 64) an option was added for different on screen character sets. This, however, works on your T.V., not your printer.

Another spell checker is available from Upstart Publishing (Dept. G3, POB 22022, Greensboro, N.C. 27420) for $15. Also available from this same place is SpeedPak. This SpeedScript expander allows for alternate screens, macros, file encryption, help screens, different character sets, mail merge, DVORAK keyboard, and three more printer codes (for use with the mail merge options). This program is also $15.

Another SpeedScript enhancement program is available from Lidon Enterprises (POB 773, Elm Grove, WI 53122). This enhancement allows the following functions:

PowerPrint – set the cursor where you want to start printing and print one character, one sentence, one paragraph or the entire document.

PowerParams – assign multiple printer commands to one key. These parameters (macros) are saved with your document for future use.

PowerSet – allows all Commodore compatible printers to switch between both character sets.

This program is available for either $4.98 for the typed listing, or $7.98 on disk or cassette.

I don't mean to criticize your magazine in any way. Rather, I am a SpeedScript enthusiast who wishes to add further insight into the SpeedScript series. Thank you for providing one of the best Commodore magazine on the market.

Jon S. Boland, Park Falls, WI

Hex – An Evil Spell: I was very amused by the letter titled 'The Horror of Hex' in the May issue of The Transactor. In it, an exasperated reader flamed that Hex 'is utter nonsense.'

My amusement was, however, bitter–sweet. I too remember first encountering hexadecimal notation. My first attempts at learning machine language involved looking up the commands in the Programmer's Reference Guide, finding their Hex opcodes, translating them to decimal and POKEing the damned creatures in! Naturally, this was not exceptionally

efficient. Soon I DESPISED Hex, and I observed that the word Hex was defined by my dictionary as 'An evil spell'!

I have often heard people voice the opinion that hexadecimal is somehow 'unnatural' and that decimal numbers are an inherent property of our universe! In my enlightenment, however, I have come to wish that humans all had 16 fingers.

Meanwhile, for those who do not wish to delve into the elegant world of machine language, there are a number of solutions. One is to use Basic extensions such as TransBASIC and MetaBASIC (published in April 85 Compute!s Gazette), which features Hex to decimal conversion commands. Another is to get a good memory map. I use 'Mapping the Commodore 64' by Sheldon Leemon, which features both hexadecimal and decimal equivalents for each address.

As for the 'private clique' of us who program in ML, well, we can use wonderful monitors such as Micromon and assemblers such as PAL. Using these beauties, we don't need to convert. Besides, what does the number 4096 signify? It's just an address, whether expressed as 4096 or $1000.

Now OCTAL . . .. That's unnatural!

Nick Barrowman, St. John's, Newfoundland

Imagine, for a moment, having 16 fingers. In that same moment something tells me a byte would have 10 bits.

The Copy Blues: I am a subscriber to The Transactor and find many good things that are passed on to our Users group here in Germany.

At the present time we are using two of the new SFD1001 drives in our BBS, and find the capacity terrific!!! However, the different formats present unique problems when it comes to the maintenance on board. We cannot find a file copy program that will transfer REL files, either to or from the 1541 and SFD. We would also like to find a full disk copy program that would make copies of the full disk from one SFD to the other SFD, so far with no success.

We realize that you have better things to do than worry about our problems, however, with the increased use of the SFD, this cannot be an isolated issue.

As a NATO organization here, we have several Canadian members in our group. They have said that if anyone can help, the Transactor can!!!! We surely hope so.

Major R.H. Jacquot
Secretary/Treasurer:
Commodore Computer Users Group Heidelberg

Although this letter was sent to us back in October of '85, it took till now to be able to reply. His request is not unique in that we have received many others like it. The drive types have varied from the SFD1001 as above through the MSD and Indus drives. One underlying factor always prevails; relative files are a drag to copy using a single disk drive.

After many months of trying to find the time to write a relative file copy program, we can now present Transcribe 64 in this issue. Transcribe 64 is a relative file copy program for the Commodore 64 that allows you to copy REL files of any length using a single drive, to and from any Commodore or compatible drive that you can interface to your 64. The only limitation in using IEEE drives with Transcribe is the interface you use. If the interface consumes RAM in the same places that Transcribe does, problems will develop. The source for Transcribe has not been printed this issue, but it does exist on the Transactor Disk #13 for all so inclined. If you find conflicts in using Transcribe with your interface, you can either re-write the source, providing you know where the conflicts have developed, or purchase a transparent interface. Your choice.

Vic Time: I own a Vic 20 (good old Vic!), 1541, a home-made 24k expansion and a printer with a Centronics port. They told me that you "couldn't" do that without an interface that would have cost me several hundred dollars, so I wrote my own interface and put it in the cassette buffer driven through the output vector. The only hardware is the cable.

In your editing please remember that there are a lot of us Vic owners around who have this queer idea that the Vic we've got is still able to do all the things we want it to do, so why spend $400 (in NZ) for a C64? I have recast programs like the R65C02 Assembler to work on the Vic screen and with my printer. It's a great program. Now I just have to get the Unassembler going, and adjust to produce code that can be handled by the Assembler.

I'm always glad to see 'translation' done for me. Like the Profiler. I was still getting around to doing it. Much simpler to see it all tested and in print. Keep it up.

Any chance of TransBASIC kernal for us Vic owners?

An idea for a TransBASIC module. How about SORT, SORT%, SORT$ which will take two elements from a numerical, integer or string array respectively and put them in the right order. Eg. SORT,a$(i),a$(i + 1). Since this is definitely a 'speed' function, it helps if the programmer tells the interpreter what sort of array is involved and not make it use up time finding out for itself. Hence three functions. For completion, there could also be an UNSORT that reverses the order regardless.

We've been getting tantalizing tidbits on the 1541 Inner Space but not enough to do to be able to do 'serious' programming in it yet. Now that we have a 1541 map of all the functions, how about some articles telling us some details of what to do with them. Richard T. Evers tells us it is done by putting the different job numbers on the JOB QUE, warns us of the dire consequences of not setting up RAM pointers correctly, and leaves it at that. Some people out there are reprogramming the drive, so the information is available. Who has it ??

Terry Montgomery, Aukland, New Zealand

The problems that we have always had with the Vic are two-fold. First, the screen is only 22 characters wide. This is a real limitation when writing software that is universal for the Commodore machines. 80 columns is just right, 40 columns is often times a pain and 22 columns is always a major drag.

Second, although the Vic is A1 for nice ROM routines and RAM vectors, it is always unpredictable when writing code to work with a specific memory configuration. The Vic was one of Commodore's initial success stories, but it was far from perfect. We have printed code written and modified for the Vic in past, and will continue to do so in the future, but without one standard memory configuration that we know is common among our readers, we can never make the Vic a habit. We just have to continue printing the source as often as possible, and hope that people such as you can make the conversions. Sorry about that.

As for programming the drive, one really terrific book exists that details all you will ever need to know about programming the 1541. The book name is 'Inside Commodore DOS', written by Richard Immers and Gerald C. Neufeld, published by Bradey, ISBN 0-88190-366-3. You will not find a better book to help you understand the inner workings of your drive. I hope this helps.

1200 Bits Per Second per second: I am glad that my letter generated some mail and more information was gathered from the discussion that ensued.

However, I think that Mr. Harsch was a little over-critical in his treatment of my letter.

I know very well that 1200bps modems work with the C-64, as I use a Volksmodem 12 modem all the time. But maybe Mr. Harsch would listen to Steve Punter in his explanation of how he discovered the secret of running 1200bps on a C-64.

The following is paraphrased from Mr. Punter's explanation of how he modified and successfully wrote the necessary code to run 1200bps from within his 'famous' BBS program.

When Mr. Punter tried to open an RS-232 channel at 1200bps by using OPEN 2,2,0,CHR$(8)+CHR$(0), he found that he could send to the modem with no problems (which goes along with Mr. Harsch's method of driving RS-232 printers). However, receiving was not so easy; he got nothing but garbage!

But he discovered that by poking different values to $0295/96, he was able to fine tune the receive timing values and make 1200bps work with his BBS program. If you look at the Kernal routines to open an RS-232 channel, the bytes are loaded into locations $0293 to $0296. But if the lower nybble of the control register is not zero, the Kernal goes to the lookup table and loads the values found there into bytes $0295/96.

Some of the more advanced terminal programs around give you the option of adjusting the values at $0295/96 so that you can use the value that works best with your modem.

On reference to the PRG (Programmers Reference Guide), I submit from page 349 the second paragraph after the heading 'OPENING AN RS-232 CHANNEL':

' Up to 4 characters can be sent in the filename field. The first two are the control and command register characters; the other two are reserved for future system options.'

Most people I have talked to personally and myself have taken that to mean that the 3rd and 4th bytes of the filename are either ignored or overwritten. Other sources of 64 information have stated the same thing. Take the book 'Anatomy of the C-64' from Abacus Software. On page 28, the table they show to determine the lower nybble of the control byte specifically states that the User Defined Baud Rate option is not implemented.

In the first edition of Sheldon Leemon's book 'Mapping the 64', he also continued the same myth. However, in subsequent editions that error was corrected.

I agree that the PRG tells us how to calculate the User Defined Baud Rate but the above quoted statement usually keeps most people from trying, until they know better.

One more proof for the problem I found was with Firstterm5 by Tom Hughes. It is a fine Punter protocal term program. I used it a lot at 300bps. But when I upgraded to 1200bps, I found that it did not work at 1200bps. I received nothing but garbage (ala Mr. Punter).

The program is written in ML and I got out my favourite monitor and started to poke around in it. I found that Mr. Hughes used 2 byte filenames to open his RS-232 channels. He had options for all the speeds listed in the PRG (50-9600 BAUD).

I was able to to find room here to eliminate all the speeds except 300 and 1200bps. I used a 4 byte filename and used the value of CHR$(57) and CHR$(1) for the optional baud rate. Lo and behold, it worked!

I again hope that this helps to clear up a few cloudy points on this issue.

Lyle R. Giese, Woodstock, IL

Left Wing Interference Revisited #1: I reference to Jack Ryan's 'Left Wing Interference' letter in The Transactor Volume 6, Issue 06.

Before I purchased a 1702 monitor, I used a monochrome Zenith monitor, model ZVM-123. The Operating Guide contains the following cryptic sentence:

Note: For best results, keep the Disk Units away from the left side of the Monitor (facing screen).

This might refer to the same type of gremlins you saw mentioned in the GT-4 cartridge manual.

Jack's problem reminded me of something that happened to me with my Datasette when I first got my Commodore 64.

I'd just moved my equipment – C–64, Datasette and old black–and–white TV that served as monitor – from the dining room table to the family room, and plunked everything down on a folding metal picnic table that we'd had for years.

Before long I noticed that I was getting an inordinate number of tape I/O errors when trying to load programs. I immediately assumed the Datasette was too close to the TV, so I moved it away.

This had absolutely no effect at all, so I moved it again, separating the two even more. Still no improvement.

I eventually reached the point where the TV and Datasette were on opposite ends of the table – eight feet apart – and still the errors came. Things were getting frustrating now.

One Saturday morning I resolved to sit down at the computer and not get up until I had solved the mystery. And so I started experimenting with various configurations.

Some time in the early afternoon I noticed that I could eliminate the errors completely if I held the tape unit in my hand, at least a foot above the surface of the table. Proximity to the TV didn't seem to matter, as long as the Datasette wasn't actually touching the table top.

I decided that the table top was conducting the television's magnetism, so I tried insulating the Datasette. First I tried resting it on three issues of Compute! magazine (the old fat ones). That seemed to help a little, but I still had too many errors. A slab of pine shelving left over from an old project was next. That didn't help either.

It struck me somewhere along the way that what I needed was not just insulation from the table but physical separation from it. So my next step was to place the Datasette on an 8" x 14" block of white styrofoam, the kind used to protect microcomputers during shipping. That did it: the errors finally stopped.

What caused the tape errors? I never did figure it out completely, although I suspect that the table had actually, to some degree, become magnetized. I don't recall that the problems occurred when I first moved on the the metal, but only after a few weeks use.

Would this also happen with a disk drive? I don't know, because about the same time I got a 1541 I got a wooden computer desk, and never saw the problem again.

Frank Figlozzi, Bowie, MD

Left Wing Interference Revisited #2: Reading the letters about 'Left Wing Interference' in the May '86 issue reminded me of a very similar experience. I have been using a C–64 since late '82 with an old industrial B/W Electrohome monitor. The image waved gently at the edges. I blamed the C–64 and bought a new Zenith CRT. The waves were less but still there. I could have lived with this imperfect image if it were not for the fact that my 1541 drive suddenly failed to read program code that previously loaded perfectly. The drive was indeed sitting at the LEFT of the new monitor.

Because the Electrohome CRT was totally enclosed in a grounded metal case, and the Zenith CRT had only a plastic enclosure, I suspected that stray electromagnetic fields from the CRT were disturbing the operation of the disk drive. This gave me the key to testing my hypothesis and finding a simple cure. A large ungrounded metal plate of 16x10 inches between the Zenith and the drive eliminated the load errors. So the cure was simply a good shield. An unsightly metal plate was not a practical solution. And I was not willing to reshuffle all the carefully positioned hardware on the computer desk. There are two alternative means of shielding:

1. Eliminate the source of the radiation by coating the inside of the monitor case with a conductive paint which is then grounded.
2. Keep radiation out of the drive with the coating inside the case.

Unfortunately, this method is not available to most of us. Early Apple computers were treated this way for similar reasons.

Some time ago, I built a small wooden telephone box that straddled the drive unit from side to side, while leaving about one inch of air space at the top. With some aluminum foil from the kitchen, contact cement and duct tape, I 'coated' the inside of the box to form a shield. After attaching a grounding wire, I found that the box when placed in its usual position over the drive, provided the necessary protection. I have not had a single load error since. The Electrohome has now given way to an equally 'leaky' 1701 colour monitor. The problem recurs the moment I take the box away. The flyback transformers in both CRTs are in fact located at the left side. I do not recommend application of the plastic case. It is not possible to get a bond that will stand up to the temperature swings over time. You don't want any short circuits either.

One of my friends took his drive back to the store for an exchange at least twice. I asked him what monitor he was using, and where it was sitting. He had put the drive on TOP of the monitor. Of course there was nothing wrong with the drive once he moved it to the right of the CRT. As a parting remark, similar load errors show up with tape transfers if the Datasette is held towards the left side of the monitor. Relocating it to the right does the trick. Guess where my tape unit has been all along until I moved it?

John C. Hollemans, Oakville, Ontario

TransBASIC Installment #10

Nick Sullivan
Scarborough, Ont.

TransBASIC Notes

If you have purchased The TransBASIC Disk, a couple of minor bugs have been found in the SYMASS 3.1 Assembler. See the article later this issue titled "Save SYMASS Symbols" for details.

TransBASIC has been a regular Transactor feature for almost two years. Those who have been following the series know all about it. Recently, however, we've received letters to the effect of "what is TransBASIC?". Quite simply, TransBASIC is a method of adding new commands to BASIC (see "Part 1:" below). The commands come in 'modules' which may contain one or more commands OR functions. After merging the modules of your choice, the entire lot is assembled and linked into BASIC. The new commands can then be used just like any of the other commands that are already in the BASIC ROM when the C64 is powered up.

The TransBASIC Disk

The TransBASIC Disk contains all of the modules published so far and it comes with its own assembler, SYMASS 3.1. Any combination of modules can be linked into BASIC with only a few simple steps. From start to finish is usually no more than a couple of minutes. . . even less once you get the hang of it. It comes with a handy reference for just $9.95. See the order card at center page.

TransBASIC Parts 1 to 8 Summary:

Part 1: *The concept of TransBASIC – a custom command utility that allows one to choose from a library only those commands that are necessary for a particular task.*

Part 2: *The structure of a TransBASIC module – each TransBASIC module follows a format designed to make them simple to create and "mergeable" with other modules.*

Part 3: *ROM routines used by TransBASIC – many modules make use of ROM routines buried inside the Commodore 64. Part 3 explains how to use these routines when creating new modules.*

Part 4: *Using Numeric Expressions – details on how to make use of the evaluate expression ROM routine.*

Part 5: *Assembler Compatibility – TransBASIC modules are written in PAL Assembler format. Techniques for porting them to another assembler were discussed here.*

Part 6: *The USE Command – The command 'ADD' merges TransBASIC modules into text space. However, as more modules are ADDed, merging gets slow. The USE command was written to speed things up. USE also counts the number of statements and functions USEd and updates the totals (source line 95) automatically.*

Part 7 – *Usually TransBASIC modules don't need to worry about interfering with one another. When two or more modules want to alter the same system vector, however, a potential crash situation exists. Part 7 deals with avoiding this problem.*

Part 8 – *Describes the five modules for Part 8.*

Part 9 – *Describes the six modules for Part 9, and makes first mention of The TransBASIC Disk.*

TransBASIC Installment #10

Time presses extremely this issue, so I'll do little more here than briefly introduce the six short modules below. First, though, one matter that can't wait.

John Houghton of Collingwood, Ontario, has drawn our attention to a bug in the MC GRAPHICS module of TransBASIC Installment #8 (volume 6, number 6). The bug consists of several lines that I inadvertently transposed in editing the code. To fix it, load in the source code, and type the following lines:

```
11080 lda $d018
11082 sta mcuvid+2
11098 lda #$68
11100 sta $d018
```

A bit later in the same module, a dollar sign found its way into the following line, from which it must be deleted (unlike the previous one, this bug would have been reported as a syntax error when you assembled the module):

```
11264  stx t5
```

I apologize to anyone who has had trouble because of the above errors.

Mr. Houghton also points that "When returning to the original text screen, sometimes the screen colours will be mangled. This is because the colour RAM is not stored when entering and exiting MC GRAPHICS. A new module would be needed to take care of this, or perhaps use MOVE & FILL. This leads to the next bug. FILL works fine, but beware, as written, MOVE does not. The bug is in lines 8200 to 8202. The destination address is not changed from Floating Point notation to Integer. Add line 8201 as:

```
8201 jsr $b7f7 ;fp1 to int
```

With this fix, one can preserve colour RAM with MOVE prior to entry, and MOVE it back on exit."

One extremely trivial bug occurs in the PRG MNGMNT module. Line 11 is a only a remark, but the REM was left off. If this were the last module USEd, assembling would stop with a ?SYNTAX ERROR. Simply insert the "REM" at the beginning of line 11 (or delete line 11 if you like), and RUN it again. This has been fixed on The TransBASIC disk.

This issue's modules are LWRITE (Program 1), by Steve Hammer of Muscatine, Iowa; RESTORE (Program 2) and CLRA (Program 3), both by past contributor Wayne Happ of North Babylon, New York; CCMDS (Program 4), by Joel M. Rubin of San Francisco, California, also a previous contributor; SPEED (Program 5), by Guido Struben of Calgary, Alberta; and TRAP (Program 6), by William Turner of Winnipeg, Manitoba.

Several of these modules alter system vectors, which has meant that the PAL-specific assembler directive:

```
.IF >(*&255) + 1: * = * + (*&1)
```

was needed on a few occasions. Users of Robert Huehn's SYMASS assembler should change these lines to:

```
.PAD
```

when typing the modules in. This change will also be required in the MC GRAPHICS module at line 12150 and the PRG MNGMNT module at line 182. Once again, if you have The TransBASIC disk, these changes have already been made.

Well, as I said, things are a trifle hasty this time around, so that's all for now. See you next time, I hope at greater length.

New Commands

LDVN (Type: Statement Cat #: 161)
Line Range: 12190–12210
Module: LWRITE
Example: LDVN 5
This statement sets the printer device number for the LWRITE and LLIST commands contained in this module. Device numbers 4 through 7 are legal. The default is 4.

LSEC (Type: Statement Cat #: 162)
Line Range: 12212–12218
Module: LWRITE
Example: LSEC 7
This statement sets up a secondary address to be used by the LWRITE and LLIST commands contained in this module. The allowed range is 0 through 255. The default is 255.

LWRITE (Type: Statement Cat #: 163)
Line Range: 12212–12314
Module: LWRITE
Example: LWRITE " A PERMANENT RECORD "
Example: LWRITE A,B$,C(3) + 2;
This statement is almost exactly the same as the standard PRINT statement, except that output is directed to the serial device specified with the LDVN command in this module (default 4). The command eliminates the need for opening and closing a file to the printer. The keyword LWRITE was used instead of the standard LPRINT in order to avoid a collision with TransBASIC's LP command, which appeared in the SOUND THINGS module.

LLIST (Type: Statement Cat #: 164)
Line Range: 12212–12314
Module: LWRITE
Example: LLIST
Example: LLIST 37–150
This statement is the same as the standard LIST statement, except that output is directed to the serial device specified with the LDVN command in this module (default 4). The command eliminates the need for opening and closing a file to the printer, and for using the BASIC CMD command to divert output there.

RESTORE (Type: Statement Cat #: 165)
Line Range: 12316–12344
Module: RESTORE
Example: RESTORE
Example: RESTORE 100
This statement is the same as a BASIC RESTORE, but accepts a target line number if one is provided. Subsequent data reads will begin from the target line.

CLRA (Type: Statement Cat #: 166)
Line Range: 12346–12356
Module: CLRA
Example: CLRA

This statement eliminates any arrays that have been created, without affecting regular variables.

CGOTO (Type: Statement Cat #: 167)
Line Range: 12358–12362
Module: CCMDS
Example: CGOTO 100 + 10*N
The target line number in this variant of the GOTO statement is calculated from the argument, which can be any valid BASIC expression.

CGOSUB (Type: Statement Cat #: 168)
Line Range: 12364–12394
Module: CCMDS
Example: CGOSUB SR
The target line number in this variant of the GOSUB statement is calculated from the argument, which can be any valid BASIC expression.

CRUN (Type: Statement Cat #: 169)
Line Range: 12396–12404
Module: CCMDS
Example: CRUN –100*(M = 4)
The target line number in this variant of the RUN statement is calculated from the argument, which can be any valid BASIC expression. Unlike RUN itself, CRUN cannot be used alone -- the argument expression is required.

CRESTORE (Type: Statement Cat #: 170)
Line Range: 12406–12446
Module: CCMDS
Example: CRESTORE 25
This version of the RESTORE statement requires an argument expression, which specifies the line from which data is to be read by subsequent READ statements.

SPEED (Type: Statement Cat #: 171)
Line Range: 12448–12542
Module: SPEED
Example: SPEED 15
This statement specifies a speed at which all printing will be performed. Larger arguments slow printing, to a maximum of 255. A zero argument restores normal-speed printing.

TRAP (Type: Statement Cat #: 172)
Line Range: 12544–12664
Module: TRAP
Example: TRAP 100
This statement specifies a line at which program execution will resume if an error, such as a syntax or illegal quantity error, should occur. The only other effect on the program is that the stack is cleared, ending any current FOR–NEXT loops and escaping to top level from any currently–executing subroutines.

ERROR (Type: Function Cat #: 173)
Line Range: 12666–12670
Module: TRAP
Example: IF ERROR = 11 THEN PRINT " BAD SYNTAX! "
This pseudo–variable contains the BASIC error number of the most recently trapped program error.

ERRLIN (Type: Function Cat #: 174)
Line Range: 12672–12692
Module: TRAP
Example: PRINT " THE ERROR WAS IN LINE " ;ERRLIN
This pseudo–variable contains the number of the line in which the most recently trapped program error occurred.

Program 1: LWRITE

```
LK  0 rem lwrite (s. hammer 12/85)      :
FH  1 :
JH  2 rem 4 statements, 0 functions
HH  3 :
DF  4 rem keyword characters: 19
JH  5 :
NJ  6 rem keyword     routine   line     ser #
BL  7 rem s/ldvn      ldv       12190    161
BF  8 rem s/lsec      lse       12212    162
FN  9 rem s/lwrite    lprn      12236    163
CC  10 rem s/llist    lst       12252    164
PH  11 :
KD  12 rem ================================
BI  13 :
FK  149 .asc "lwritEldvNlseCllisT"
NH  1149 .word lprn-1,ldv-1,lse-1,lst-1
EI  12190 ldv   jsr  $b79e    ;get device #
NK  12192       cpx  #3       ;test 3<=dv<=8
BJ  12194       bcc  badev    ; no
NA  12196       cpx  #8
FN  12198       bcs  badev    ; no
KJ  12200       stx  dv
GJ  12202       rts
CC  12204 ;
IH  12206 badev ldx  #9       ;'illegal device #'
AL  12208       jmp  $a437
IC  12210 ;
KI  12212 lse   jsr  $b79e    ;get sec address
FJ  12214       stx  se
EK  12216       rts
AD  12218 ;
BL  12220 opf   jsr  lpr1     ;ensure file closed
JL  12222       lda  #0       ;no name
LJ  12224       jsr  $ffbd    ;kernal setnam
CA  12226       lda  #$78     ;file # 120
IH  12228       ldx  dv
LG  12230       ldy  se
JK  12232       jsr  $ffba    ;kernal setlfs
OC  12234       jmp  $ffc0    ;kernal open
CE  12236 ;
LA  12238 lprn  jsr  opf      ;open print file
BO  12240       ldx  #$78     ;file #
EO  12242       jsr  $79
```

```
JK | 12244          jsr  $aa90     ;print rtn entry
AH | 12246 lpr1     jsr  $abb5     ;basic clrchn
HH | 12248          lda  #$78      ;close file
CE | 12250          jmp  $ffc3     ;kernal close
CF | 12252 ;
PF | 12254 lst      jsr  opf       ;open print file
OG | 12256          jsr  lfix      ;grab warmstt vec
DP | 12258          ldx  #$78      ;file #
FA | 12260          stx  $13
FG | 12262          jsr  $e118     ;basic chkout
EF | 12264          jsr  $aad7     ;print cr & lf
PH | 12266          jsr  $79       ;reget byte
PP | 12268          jmp  $a6a4     ;basic list entry
EG | 12270 ;
DA | 12272 lfix     lda  $0300     ;direct warmstart
FG | 12274          ldy  $0301     ; vector to
NG | 12276          sta  wsvec     ; routine below
JD | 12278          sty  wsvec+1
CA | 12280          lda  #<lret
AG | 12282          ldy  #>lret
PK | 12284          sta  $0300
CB | 12286          sty  $0301
MO | 12288          rts
IH | 12290 ;
CO | 12292 lret     lda  wsvec     ;restore old
LI | 12294          ldy  wsvec+1   ; ws vector
LL | 12296          sta  $0300
OB | 12298          sty  $0301
IH | 12300          jsr  $aad7     ;print cr & lf
LG | 12302          jsr  lpr1      ;clrchn, close
PB | 12304          jmp  $e386     ;exit to ready
II | 12306 ;
JD | 12308 dv       .byte 4        ;print dev #
MK | 12310 se       .byte $ff      ;print sec addr
KA | 12312 wsvec    .word $e38b    ;warmstart rtn
AJ | 12314 ;
```

Program 2: REST

```
IA | 0 rem rest (w. happ)          :
FH | 1 :
AI | 2 rem  1 statement, 0 functions
HH | 3 :
HO | 4 rem keyword characters: 7
JH | 5 :
NJ | 6 rem keyword     routine     line     ser #
KO | 7 rem s/restore   res         12316    165
MH | 8 :
OO | 9 rem ================================
OH | 10 :
LL | 150 .asc "restorE"
PK | 1150 .word res-1
BF | 12316 res     beq  re2       ;no parameter
FN | 12318         jsr  $a96b     ;ascii to integer
LP | 12320         jsr  $a613     ;test line exists
IM | 12322         bcc  re3       ; no
EE | 12324         ldx  $5f       ;get line address
OA | 12326         ldy  $60
KF | 12328         dex            ;point to null byte
GE | 12330         bne  re1       ; before line
HO | 12332         dey
LF | 12334 re1     stx  $41       ;update data ptr
IF | 12336         sty  $42
OB | 12338         rts
IG | 12340 re2     jmp  $a81d     ;basic restore
FG | 12342 re3     jmp  $a8e3     ;'undef statement'
OK | 12344 ;
```

Program 3: CLRA

```
AN | 0 rem clra (w. happ)          :
FH | 1 :
AI | 2 rem  1 statement, 0 functions
HH | 3 :
EO | 4 rem keyword characters: 4
JH | 5 :
NJ | 6 rem keyword     routine     line     ser #
PI | 7 rem s/clra      clr         12346    166
MH | 8 :
OO | 9 rem ================================
OH | 10 :
EM | 151 .asc "clrA"
BO | 1151 .word clr-1
PM | 12346 clr      lda  $2f       ;set end-of-arrays
IO | 12348         ldy  $30       ; ptr to start-of-
LM | 12350         sta  $31       ; arrays
GG | 12352         sty  $32
OC | 12354         rts
KL | 12356 ;
```

Program 4: COMPUTED CMDS

```
IB | 0 rem computed cmds (j. rubin sept/85):
FH | 1 :
JH | 2 rem 4 statements, 0 functions
HH | 3 :
IE | 4 rem keyword characters: 23
JH | 5 :
NJ | 6 rem keyword     routine     line     ser #
MD | 7 rem s/cgoto     cgto        12358    167
KH | 8 rem s/cgosub    cgsb        12364    168
CP | 9 rem s/crun      crn         12396    169
BP | 10 rem s/crestore crsr        12406    170
PH | 11 :
KD | 12 rem ================================
BI | 13 :
CP | 152 .asc "cgotOcgosuBcruNcrestorE"
CI | 1152 .word cgto-1,cgsb-1,crn-1,crsr-1
PO | 12358 cgto    jsr  clin      ;eval line #
NP | 12360         jmp  $a8a3     ;enter goto
AM | 12362 ;
GC | 12364 cgsb    lda  #3        ;check stack space
ML | 12366         jsr  $a3fb
KE | 12368         lda  $7b       ;push chrget ptr
OM | 12370         pha
BB | 12372         lda  $7a
```

```
CN | 12374          pha
LA | 12376          lda  $3a        ;push line #
GN | 12378          pha
JP | 12380          lda  $39
KN | 12382          pha
MG | 12384          lda  #$8d       ;push gosub token
ON | 12386          pha
GH | 12388          jsr  $79
KE | 12390 cgs1     jsr  cgto       ;"crun" entry
ND | 12392          jmp  $a7ae      ;interpreter loop
AO | 12394 ;
AI | 12396 crn      lda  #0         ;kernal msgs off
JN | 12398          jsr  $ff90
PN | 12400          jsr  $a660      ;clr
EH | 12402          beq  cgs1       ;branch always
KO | 12404 ;
HN | 12406 crsr     jsr  clin       ;eval line #
PG | 12408          lda  $7a        ;push chrget ptr
GP | 12410          pha
MD | 12412          lda  $7b
KP | 12414          pha
MO | 12416          jsr  $a8a3      ;basic goto
HK | 12418          sta  $42        ;new chrget ptr to
OG | 12420          lda  $7a        ; data pointer
LE | 12422          sta  $41
CH | 12424          pla             ;restore chrget ptr
II | 12426          sta  $7b
EB | 12428          pla
JI | 12430          sta  $7a
ND | 12432 cre1     rts
IA | 12434 ;
GF | 12436 clin     jsr  $ad8a      ;eval num expr
CC | 12438          jsr  $b7f7      ;conv to integer
AN | 12440          cmp  #$fa       ;test < 64000
FH | 12442          bcc  cre1       ; yes
OD | 12444          jmp  $b248      ;'illegal qty'
EB | 12446 ;
```

Program 5: SPEED

```
JM | 0 rem speed (guido strben 11/85)    :
FH | 1 :
AI | 2 rem  1 statement, 0 functions
HH | 3 :
FO | 4 rem keyword characters: 5
JH | 5 :
NJ | 6 rem keyword     routine    line     ser #
OJ | 7 rem s/speed     spd        12448    171
MH | 8 :
OO | 9 rem ================================
OH | 10 :
NK | 153 .asc "speeD"
JJ | 1153 .word spd-1
IC | 2125          jsr  morvec
EL | 2555          jsr  kilvec
NF | 9162 morvec =    *
IO | 9165          jsr  spdon      ;print $326
GM | 9178          rts
CF | 9180 ;
```

```
MJ | 9182 kilvec =    *
AJ | 9185          jsr  spdoff     ;print $326
KN | 9198          rts
GG | 9200 ;
BK | 12448 spd     jsr  $b79e      ;get byte in .x
EK | 12450          stx  xbyte      ;put it away
KB | 12452          cpx  #0         ;test speed 0
KA | 12454          beq  spdoff     ; yes
DI | 12456          lda  $0326      ;test vec altered
EK | 12458          cmp  prvec
BK | 12460          beq  sp1        ; no
CD | 12462          lda  $0327
BJ | 12464          cmp  prvec+1
II | 12466          bne  sp3        ; yes
OG | 12468 sp1      lda  #<dly      ;point to delay
FF | 12470          ldy  #>dly      ; routine
IG | 12472 sp2      sta  $0326
MN | 12474          sty  $0327
BB | 12476 sp3      rts
ED | 12478 ;
MG | 12480 dly      pha             ;push registers
GH | 12482          txa
AE | 12484          pha
NH | 12486          tya
EE | 12488          pha
IG | 12490          ldx  xbyte      ;get speed
BK | 12492 dl1      ldy  #$c8       ;init counter
MI | 12494 dl2      dey             ;eat time
EM | 12496          bne  dl2
JI | 12498          dex
FM | 12500          bne  dl1
PL | 12502          pla             ;restore registers
HK | 12504          tay
CG | 12506          pla
HK | 12508          tax
GG | 12510          pla
MH | 12512          jmp  (prvec)    ;print
IF | 12514 ;
OJ | 12516 spdon    lda  $0326
KM | 12518          ldy  $0327
LN | 12520          sta  prvec
EC | 12522          sty  prvec+1
IN | 12524          rts
EG | 12526 ;
KB | 12528 spdoff   lda  prvec
OO | 12530          ldy  prvec+1
PP | 12532          bne  sp2
MG | 12534 ;
BN | 12536 .if >(*&255)+1: *=*+(*&1)
BG | 12538 prvec    .word 0
GN | 12540 xbyte    .byte 0
EH | 12542 ;
```

Program 6: TRAP

```
NM  0 rem trap (william turner)        :
FH  1 :
EI  2 rem  1 statement, 2 functions
HH  3 :
LE  4 rem keyword characters: 15
JH  5 :
NJ  6 rem keyword     routine     line     ser #
BJ  7 rem s/trap      tra         12544    172
ME  8 rem f/error     erro        12666    173
CK  9 rem f/errlin    errl        12672    174
OH  10 :
JD  11 rem ================================
AI  12 :
NO  154 .asc "traP"
KC  623 .asc "erroRerrliN"
GJ  1154 .word tra-1
II  1623 .word erro-1,errl-1
IC  2125          jsr   morvec
EL  2555          jsr   kilvec
NF  9162 morvec   =     *
NF  9164          jsr   auton       ;warm start $302
KE  9166          jsr   erron       ;error $300
GM  9178          rts
CF  9180 ;
MJ  9182 kilvec   =     *
ID  9184          jsr   autoff      ;warm start $302
CC  9186          jsr   erroff      ;error $300
KN  9198          rts
GG  9200 ;
JM  9202 autoff   lda   wrmsrt      ;from d. spruyt's
IO  9204          sta   $302        ; prg management
GN  9206          lda   wrmsrt+1    ; module
EI  9208          sta   $303
GO  9210          rts
CH  9212 ;
ND  9214 auton    lda   $302        ;from d. spruyt's
EF  9216          sta   wrmsrt      ; prg management
ON  9218          lda   $303        ; module
EJ  9220          sta   wrmsrt+1
CP  9222          rts
GI  9232 ;
LD  10082 .if >(*&255)+1: *=*+(*&1)
KC  10084 wrmsrt .word 0
MN  10086 ;
MF  12544 tra     jsr   $ad8a       ;get addr for trap
DH  12546         jsr   $b7f7
JJ  12548         sty   tlin        ;store line #
HL  12550         sta   tlin+1
PJ  12552         tax               ;test trap off
PM  12554         bne   tra1        ; no
DM  12556         tya
CJ  12558         beq   erroff      ; yes
MI  12560 tra1    bit   trflag      ;test trap on now
MC  12562         bmi   tra2        ; yes
IE  12564         lda   #<newv      ;use new err vec
BK  12566         ldy   #>newv
LM  12568         sta   $0300
OC  12570         sty   $0301
IE  12572         lda   #<warm      ;use new ws vector
AG  12574         ldy   #>warm
FN  12576         sta   $0302
ID  12578         sty   $0303
JG  12580         lda   #$80        ;trap flag on
FJ  12582         sta   trflag
IO  12584 tra2    rts
AK  12586 ;
FB  12588 newv    bmi   warm        ;branch on run-stop
DK  12590         stx   errno       ;store error #
NF  12592         lda   $39         ;store line #
HE  12594         ldy   $3a
PA  12596         sta   eline
IF  12598         sty   eline+1
AP  12600         bit   trflag      ;test trap on
EN  12602         bpl   eout        ; no
OI  12604         lda   tlin        ;get trap line #
BB  12606         ldy   tlin+1
IA  12608         sta   $14
NG  12610         sty   $15
IN  12612         ldx   #$fa        ;clear stack
CE  12614         txs
FK  12616         lda   #$a7        ;push interpreter
BJ  12618         pha               ; loop address
CP  12620         lda   #$ad
KM  12622         pha
JJ  12624         jmp   $a8a3       ;goto trap line
IM  12626 ;
MB  12628 warm    jsr   erroff
AG  12630         jsr   autoff
JK  12632 ;       jsr   ownrtn      ;cancel hi-res etc
JD  12634         ldx   #$80        ;exit to 'ready.'
OO  12636         jmp   ($0300)
EN  12638 ;
LL  12640 erroff  lda   errvec
OO  12642         ldy   errvec+1
HB  12644         sta   $0300
KH  12646         sty   $0301
IJ  12648         lsr   trflag      ;trap flag off
GF  12650         rts
CO  12652 ;
JA  12654 erron   lda   $300
GB  12656         ldy   $301
KB  12658         sta   errvec
OD  12660         sty   errvec+1
CG  12662         rts
OO  12664 ;
PL  12666 erro    ldy   errno
OK  12668         jmp   $b3a2
EP  12670 ;
BJ  12672 errl    ldy   eline
GA  12674         lda   eline+1
ME  12676         jmp   usfp
MP  12678 ;
CG  12680 eout    .byte $4c         ;jmp
NO  12682 errvec  .word 0           ;old error vector
IP  12684 tlin    .word 0           ;trap line
IK  12686 errno   .byte 0           ;error number
EO  12688 eline   .word 0           ;erroneous line #
OI  12690 trflag  .byte 0           ;trap set flag
KA  12692 ;
```

The ATARI ST Notebook

Jack Cole
Kitchener, Ont.

New hardware is, after all, new hardware. And using a new operating system is, after all, like opening a can of worms. . .

Sometime early in the summer of 1985 we became one of the first developers privileged to receive a pre–production version of the Atari 520ST. To say we were privileged is of course dependent on your point of view. We were all excited at the prospects of *playing* with the beast, but the thought of doing *real work* on it left us quite apprehensive. New hardware is, after all, new hardware. And using a new operating system is, after all, like opening a can of worms – with your teeth. And **pre–releases** are generally best left until they are no longer pre–releases. And pre–releases with windows and rodents are best dragged quickly over to the trash can symbol (or so we believed). The ST had tremendous potential in this regard; it could be potentially difficult to get our code where we wanted it, and it could be potentially impossible to debug. New machines very often wither away and die from these maladies as may the programmers who beat their heads against them.

As it turned out, we needn't have concerned ourselves. The ST is a refreshingly solid and simple system.

Our mission was to investigate and hopefully port our network (IMAGINET) and software products from the PC–10, Compaq, and IBM compatible machines to the new Atari, there to provide low cost and high performance network workstations for our database (The Manager). This notebook will provide an overview of the Atari programming environment and how **we** found in a development atmosphere. Please remember that our needs and desires in a computer may bear little resemblance to yours, so some of our criticisms and preferences may seem trivial to you. They probably are!

The Hardware

First of all a brief description of the ST hardware may be in order. I won't go into much detail here though; lots of articles have been written in other places about the 520 and it's new big brother the 1040. Especially see the November 1985 issue of **The Transactor** and the March 1986 issue of **BYTE** which looks inside the 1040 in very nice detail. Our experience has mostly been with the 520 without the operating system in ROM; however our recent 1040 investigations indicates that comments here apply to all versions of the machines.

The ST comes in several pieces; more on this later. Both colour and monochrome monitors are available. The monochrome, operating in high resolution mode, offers 640 by 400 pixels; the colour offers 640 by 200 with four colours (medium resolution) or 320 by 200 with 16 colours (low resolution). What this really means is that in order to view 80 columns – an absolute necessity for professional programming or business use – only four colours can be used. This turns out to be a bit restrictive for some applications, but not fatal.

The 1040 has a built–in 3.5 inch double–sided floppy drive. The external drives (you may have two drives attached) come in either single– or double–sided versions, with formatted capacities of 360k and 720k respectively. The drives seem to be slower than they should, but the real drag with the externals is that each requires a separate power supply, and operates at the end of a *very* short cord from the main system unit. As a result (especially true of the 520) the drives and paraphernalia tend to be annoyingly in the way.

Our prototype hard disks – a slight variation of SCSI disks, dubbed ACSI by ATARI – have performed well. The 20 megabyte drive is extremely quick and seems very reliable. Now that this drive is ready for the general public, it should knock several layers of socks off, and at a very affordable price!

The DMA port, through which the hard disk is connected, is intended to support up to 8 devices. Already developed and waiting for the "right time" is a CD–ROM – a very large capacity, read only storage system using compact disk technology. ATARI is supposedly waiting for a CD that can be used both for audio *and* with the ST. The DMA port is also the logical place for a network connection, and indeed is where ours connects, looking for all the world to the ST like a collection of hard disks. The two complaints we have with the DMA port are that (again) the cables must be very short, and the ST has difficulty (electrically) in handling more than one device. ATARI (or a third party) is believed to be building an eight connector box to remedy these problems.

The system unit is the real meat. Driven by a Motorola 68000 16–almost–32 bit processor, it also contains the RAM (512k or 1M), the operating system in ROM (early versions of the machine did not have TOS in ROM, but it is now available), MIDI, RS232, mouse and parallel ports, interfaces for the disk drives and monitors, power supplies for the 1040, and so on. Oh, and one other thing. The system unit is really the keyboard.

The layout of the keyboard is very similar to that of an IBM PC; undoubtedly a good bit of planning as it makes migration from that machine a bit easier. It has 10 function keys, an 18 key numeric cluster and an extra 8 key cluster, in addition to the

regular alpha key layout. The *feel* of the keyboard is, however, less than satisfactory. The keys are dead to the touch offering no tactile feedback, so that a former PC user would be very uncomfortable typing on the ST. The 1040 has made very slight improvements in this area; more are needed.

The second major complaint with the physical side of the ST is that it does indeed come in pieces. A two–floppy 520 development system comprises 3 power supplies, 4 power cords, and 3 other very short cables, apart from the 4 basic components. This results in a cluttered and awkward desk, and a difficult system to transport. The plethora of cables means that the keyboard is virtually fixed in place. The 1040, by building in two of the supplies, one floppy and an RF modulator (in some models) will make the ST much tidier and easier to pick up and take home from the office. In the mean time there's a fortune to be made selling nice little cases to pack your ST into.

The Operating System

Many 68000 based machines have been brought to market in the last 3 or 4 years, necessitating 68000 based operating systems. Most developers went to some UNIX variation or another, because UNIX had been well shaken down on various Digital Equipment machines and the 68000 suited UNIX to a tee. Besides which, it was trendy to have UNIX available; sort of like the flavour of the month at the local Yuppie Gourmet Tofu Palace. So, many 68000 boxes come with an official UNIX System, some other port (with a UNI prefix or IX suffix) or a re–implementation by another software house. Motorola, as the chip developer, got off to an early start with its own VersaDOS system and then jumped on the bandwagon by offering UNIX as an alternate system for their VME line. Commodore, as was discussed in a recent issue of **The Transactor**, came out with a new multi–tasking multi–window DOS for the Amiga. But what I believe these people missed when bringing their stuff to market, was that such offerings are much too complex for nearly everyone's needs. (Mind you, only one of these companies is truly **aiming** at the mass market!) As a one–time UNIX system's fanatic (I, too, was once guilty of porting UNIX to a new 68000 box) I can appreciate its tremendous power and efficiency. However putting UNIX–type products in front of most users, is like giving them a 747 when they asked for a Toyota. It has way too many gadgets, takes way too many resources, makes mostly noise and smoke, requires lots of special training and probably doesn't fit in the garage anyway! On the whole it creates more problems than it solves; more people will give up than will manage to take off.

Some percentage of the (briefly euphoric) latter group may actually land safely again.

The ST operating system (TOS) turned out to be a very pleasant surprise. When ATARI needed an operating system developed, they went to Digital Research (DRI) – literally; ATARI supplied most of the people and did much of the work while DRI retained the rights. Using DRI proved to be both a blessing and a curse. The curse is that much of TOS is now out of ATARI's hands, and so problems with TOS may take a long time to be fixed, simply because ATARI does not have the means and DRI does not have the inclination. Hopefully these issues will be resolved, so that future releases of TOS can correct some of the existing problems.

The blessings are that TOS has so few of these "problems", and is **not** simply "son of UNIX" or even "son of CPM" as it could easily have ended up. While there are some CPM/68K holdovers, the really good news about TOS is that it is basically (are you ready) an **MSDOS compatible system!** That's right, DRI took the foundation of MSDOS and recreated it on the ST. Note that this does not mean that programs from your PC will run on the ST. It does mean that they can be moved to the ST much more easily. While they were at it, DRI moved their **GEM** windowing environment (developed for MSDOS) as well. People familiar with windows will find GEM's Desktop very familiar (Apple believed it was much *too* familiar, and sued DRI); the trash can, disk and file icons are there, as are the standard mouse operations. The screen is quite nicely done; much cleaner than the Amiga screen. Those of you who dislike rodents will be pleased to know that the windowing environment can be *punted* (if it is in RAM), leaving a *shell* (the proper name for a Command Line Interpreter) which will execute commands from the keyboard.

The on–disk structures are **identical** to those of MSDOS (very conveniently allowing PC's and ST's to share disks on a network). The resulting filesystem is very straightforward; it is not easily damaged and quite simple to repair. The operating system calls (open a file, read a file, etc.) are nearly the same as MSDOS, making the job of software porting very direct. This was another smart move by ATARI, as it should speed up the availability of popular software packages.

However, all is not perfect. TOS has not yet been equipped with the adequate set of utilities that make up MSDOS. If you work outside of the windowed environment, you must be prepared for frustrations. The shell has a few–built in commands (COPY, RENAME, TYPE etc.) but they are awkward and do not accept wildcards properly. There are very few external commands, and they are equally crude. Fortunately, the C compiler, assembler and linker included with our development kit worked fine (well, nearly so) and we were able to surmount these problems.

The first things we moved to the ST were our own screen editor, shell, object librarian, file system patcher, copy program, and a collection of other useful utilities. Our familiar programming environment from the PC was all in place on the ST in 3 or 4 weeks, so that we now have two nearly indistinguishable systems to work on.

TOS Calls

As mentioned, the operating system calls are nearly identical to those of MSDOS version 2. Some of the more subtle things were missed, leading us to suspect that DRI didn't *really* know how the MSDOS versions performed but instead worked from a manual. A few calls have been removed and a couple of them work slightly different, but the discrepancies are very minor. All are reached by building a parameter block on the stack and issuing a special 68000 instruction, **TRAP 1.** The item at the

top of the stack is the 16–bit function number as described below, followed by the function's parameters. This parameter passing convention is the same as that used by **C** and other high level languages, and makes high level calls to TOS quite simple. Most results are returned via data register 0. Table 1 summarizes the TOS function calls, and their MSDOS equivalents. You will notice that most of the unimplemented calls are either for obsolete or obscure functions.

In addition there are (at last count) 50 TOS or BIOS variables whose memory locations are "cast in concrete". These include very useful items such as a timer tick counter, disk buffer list, magic vectors and configuration information. To examine/modify these items a program must first change to supervisor mode (TOS function 32), then change back when finished.

BIOS Too

There is also a fairly extensive set of BIOS (ROM resident) routines, for working at a level closer to the hardware. These functions are accessed in the same way as the TOS calls, but through **TRAP 13** and **TRAP 14** instructions. These functions are in no way related to the PC; compatibility does not extend down to the BIOS level. Some **hi–lights** of the BIOS functions are summarized in Table 2.

TABLE 1

# (dec.)	FUNCTION	COMMENTS
0	Terminate current process	
1	Read standard input with echo	
2	Display character to standard output	
3	Read character from serial port	
4	Write character to serial port	
5	Write character to the printer	
6	Direct console I/O – read AND write	
7	Read standard input without echo	
8	Same as 7, but check for control characters	
9	Print string to standard output	different terminating character
10	Read a line from standard input	
11	Check standard input for character ready	
12	Not implemented	MSDOS: Clear input buffer and do input
13	Not implemented	MSDOS: Flush disk buffers
14	Select default disk drive	TOS: returns more information
15	Not implemented	MSDOS: old file open
16	Check standard output ready	MSDOS: old file close
17	Check printer ready	MSDOS: old find file
18	Check serial port character ready	MSDOS: old find next
18	Check serial port character ready	MSDOS: old find next
19	Check serial port output ready	MSDOS: old file delete
20	Not implemented	MSDOS: old sequential read
21	Not implemented	MSDOS: old sequential write
22	Not implemented	MSDOS: old file create
23	Not implemented	MSDOS: old file rename
24	Not implemented	MSDOS: neither
25	Get current drive number	TOS: has 16 maximum
26	Set disk transfer address	used by function 78
27	Not implemented	MSDOS: get default disk ID
28	Not implemented	MSDOS: get specific disk ID
29	thru 31 Not implemented	MSDOS: neither
32	Toggle from user to supervisor mode	MSDOS: Not implemented
33	Not implemented	MSDOS: old random read
34	Not implemented	MSDOS: old write
35	Not implemented	MSDOS: old get file size
36	Not implemented	MSDOS: old FCB operation
37	Not implemented	MSDOS: set interrupt vector
38	Not implemented	MSDOS: old and sleazy program exec
39	Not implemented	MSDOS: old multiple record read
40	Not implemented	MSDOS: old multiple record write
41	Not implemented	MSDOS: old and sleazy filename parser
42	Get current date	format of date is different
43	Set current date	format of date is different
44	Get time	format different; TOS: poorer resolution
45	Set time	format different
46	Not implemented	MSDOS: set verify switch
47	Get disk transfer address	
48	Get version number	

Now that you know more than you probably ever expected to about the ST, let me close with some warnings. TOS and GEM are still infected with a few rather unpleasant bugs. This is not surprising; even MSDOS went through some pretty poor releases before they got it right. The people at ATARI seem very anxious to sort these things out (it is, after all, probably life or death for them) and get full marks from us for co–operation. Many of the bugs that we reported last summer and fall were fixed in the two or three updates we have since received. We were even able to visit ATARI and talk to the developers about some of the problems. Regrettably however, some rather nasty bugs have stuck around long enough to get into the first set of TOS ROMs, and thereby are assured a place in history.

One such bug allows the creation of a file when another file of the same name is already open, resulting in two files with the same name. Another prevents the use of disks with a *cluster size* (the number of sectors making up a disk allocation unit) greater than two. This turns out to be a nasty problem with networks of ST's and PC's, where such disks may already exist. The most serious known bug imposes a limit on the number of sub–directories which may be *touched* on all devices from one ST boot to the next. By "touched" I mean even **passed over** during directory listings. The limit on the 520ST is 40 sub–directories; the 1040 limit is about 15 higher. Once this limit has been reached, TOS, having exhausted a valuable resource, is unable to recover, and dies a slow death giving Out of memory errors.

The hypothesized origin of this bug is rather interesting, and actually demonstrates four individual problems. As I mentioned earlier, it appears that the TOS implementors were **not** MSDOS wizards. This shows itself in the workings of the "find first file" call, function 78. This call is often followed by multiple calls to function 79, "find next file". To allow tree searching, the calls must be recursive; that is, while doing one set of "find files" one may find a directory and branch down into another sub–tree, do more and different "find files", and then wish to continue the original set. Hence the "find file" calls must remember the environment they were working with at each

TABLE 1 (cont'd)

# (dec.)	FUNCTION	COMMENTS
49	Terminate process and stay resident	
50	Not implemented	MSDOS: neither
51	Not implemented	MSDOS: control–break check
52	Not implemented	MSDOS undocumented: get busy flag
53	Not implemented	MSDOS: get interrupt vector
54	Get disk information/free space	
55	Not implemented	MSDOS undocumented: get/set switch character
56	Not implemented	MSDOS: get country dependent information
57	Make directory	
58	Remove a directory	
59	Change current directory	
60	Create a new file	
58	Remove a directory	
59	Change current directory	
60	Create a new file	
61	Open a file	ST bug: file is always opened for writing
62	Close a file	
63	Read from a file	
64	Write to a file	
65	Delete a file	
66	Seek within a file	
67	Change file attributes (read only etc.)	
68	Not implemented	MSDOS: I/O control for devices
69	Duplicate a file handle	
70	Force a duplicate of a file handle	
71	Get current directory	TOS: works differently at root
72	Allocate memory	MSDOS: allocated in chunks of 16 bytes
73	Free allocated memory	
74	Shrink a memory block	MSDOS: can also grow a block
75	Exec another process	
76	Terminate a process and restart parent	
77	Not implemented	MSDOS: get return code. TOS: part of exec
78	Find first matching file	
79	Find next matching file	
80	Not implemented	MSDOS undocumented: switch processes
81	Not implemented	MSDOS undocumented: get current process id
82	Not implemented	MSDOS undocumented: get DOS tables
83	Not implemented	MSDOS: neither
84	Not implemented	MSDOS: get verify state
85	Not implemented	MSDOS: neither
86	Rename a file	
87	Change a file's date and time	TOS: broken

TABLE 2

13–1	Return input status of character device	
13–2	Return character from input device	returns IBM scan codes
13–3	Output a character to a device	
13–4	Read/write sectors on a device	
13–5	Get or set interrupt vector	
13–7	Get disk parameters	
13–8	Get character device output status	
13–9	Check for disk media change	
13–10	Get a list of existing disks	
13–11	Get/set status of shift, alt, control keys	
14–0	Initialize mouse handler	
14–2	Get the screen's physical base address	
14–3	Get the screen's logical base address	
14–4	Get the screen resolution	
14–5	Set the preceding 3 parameters	
14–6	Set the colour palette registers	
14–7	Change one colour in palette	
14–8	Read raw floppy device	
14–9	Write raw floppy device	
14–10	Format a track on a floppy	
14–12	Write a string to the MIDI port	
14–15	Configure RS232 port	
14–16	Get keyboard translation tables	
14–17	Get a random number	
14–20	Dump screen to printer	
14–28	Read or write registers on the sound chip	
14–32	Submit a program to the sound generator	
14–33	Get or set printer configuration	
14–38	Execute some code in supervisor mode	
14–39	Throw away the window stuff	frees up 190k if TOS in RAM

"find first". MSDOS buries this information in a user–supplied buffer, putting the onus on the caller and allowing a number of searches limited only by the number of buffers a user can afford. When the process making these calls terminates, the buffers (being part of the process) are freed automatically. TOS unfortunately keeps this necessary information in internal buffers, which it takes from a (finite sized) small pool it keeps for such purposes. This is the first problem. When the pool is used up, the second bug steps in and kills TOS with the misleading out of memory condition. Even this might be infrequent enough to be of little concern were it not for bugs three and four. Inexplicably, TOS decides to allocate one of these buffers every time it touches a sub–directory, even if it doesn't open it! Further, the buffers are not given back to the pool after a process terminates. Consequently, TOS blows up much more often than it should.

Now admittedly, 40 sub–directories is a fair lot especially for a floppy disk system. However, for a hard disk system or a multiple hard disk network this limit is a killer. One pass over our office's network mail system, and TOS blows up! Real good.

The Finish

I hope this has served as a useful "inside" introduction to the ST. If you crave additional data, there's lots more where this came from. Despite some negative remarks that may have appeared, we are all extremely positive about the ST. Many of our complaints have already been addressed by the 1040, and those remaining are more than offset by the tremendously good price of this powerful and flexible box. Besides, forewarned is forearmed. The ST is the logical next choice for those looking for more capabilities than their 8–bit home computer provides, without having to learn to fly a 747. From our perspective it is also a viable choice for professional business and programming people. I think you'll be seeing a lot of ATARI from here on in.

Jack Cole is Research and Development Project Leader at BMB CompuScience in Milton, Ontario.

Commodore 128 Disk Boot Basics

Jim Butterfield
Toronto, Ontario

You probably know that the Commodore 128 has an Autoboot feature. If you have a disk in your drive at the time you turn on the power, you may get a message such as BOOTING. . . following which a program may start automatically.

I'd like to talk about how the BOOT mechanism works; there are a surprising number of combinations built into the system. Most users have a Bootmaking program which will generate a simple autoboot.

Methods

All of the following events happen after the computer has been completely initialized. Basic is in place and everything is running normally; but just before saying "READY", Basic checks for a Boot.

1. Just after Basic starts up or when the command BOOT is given, the computer reads the contents of track 1, sector 0 from the disk into addresses 0B00 to 0BFF. Even if the disk doesn't contain a "boot" pattern, the information will be there if there's a disk in the drive. You could set up your own coding there.

2. If the first three characters on the sector are the ASCII characters "CBM", this is a Boot disk. The program will print BOOTING and we will follow the steps listed below. Otherwise, the system returns to Basic which now says READY.

3. The next four locations are checked; if they are non-zero, the program will read in "boot sectors" from track 1 (and subsequent tracks if necessary). The first two locations contain the address where the load should start. The third location specifies the bank number for the load, and the fourth tells how many blocks are to be loaded. (I have not seen this feature used on a boot disk).

4. Starting at byte 7 (the eighth byte) there may be ASCII text followed by a zero byte, or there may be just a zero byte. The system prints anything it finds before the zero byte; the text, if any, will appear behind the word BOOTING on the screen. The message BOOTING . . . is now completed with three dots. Whatever is printed need have nothing to do with any file name.

5. Starting behind the last zero (which terminated the BOOTING message), there may be ASCII text followed by a zero byte, or there may be just a zero byte. If there is any text there, it's taken to be a file name. The name is prefixed with "0:" (for drive 0), and then the computer performs a BLOAD of that file from disk. (This feature is **not** used with a disk set up with the AUTOBOOTMAKER program, even though such a disk loads and runs a Basic program named by the user).

6. Behind this second name (if any) there must be machine language. It may be nothing more than a hexadecimal 60, the RTS command to return to Basic.

Count the Ways

Let's run through the different ways we can use these mechanisms to get a program into the computer.

1. Without the CBM pattern, we'll still load track 1, sector 0; if we are sure that the code is there, we may use it even though no BOOT message is seen.

2. There may be a machine language program entirely within the block at track 1, sector 0, and we want it to run immediately. We might follow the CBM pattern with six zero bytes (four for no boot blocks, one for no message, one for no load) with the code itself, which would start running instantly.

3. There may be a machine language program entirely within the block, but we don't want it to run until called. We could use a pattern similar to that described above, except that we would prefix a RTS command (hex 60) at the start of our code. That way, the system would return to Basic at boot time, and we could call the program with a SYS when we needed it.

It would be better to give the user some help rather than have him or her trying to remember what SYS address to use. We could put the SYS command onto a function key – see the DOS SHELL detail given below (but avoid using BANK 12). Or to make it more simple, we could give the SYS address as part of the BOOT name. Here's how it would go: suppose our machine language code starts at address $40 within the block. We'd write the block as follows:

Start with ASCII CBM (hex 43 42 4D). Follow that with four zeros (no boot blocks). Follow that with the ASCII message "SYS2880" – 2880 is the decimal equivalent of hex $B40 followed by a zero; that would be 53 59 53 20 32 38 38 30 00. Now another zero to signal that no program should be loaded. Now the "immediate" machine language program; just the RTS (hex 60). When this disk boots, you will see the command BOOTINGSYS2880. . . which will tell the user what to do.

4. You could have a massive machine language program arranged on track 1 (sectors 1, 2, 3, and so on). It could even be big enough to need space of tracks 2 or 3. The four bytes right behind the CBM will tell the system to do this load.

5. In addition to all the above: You could have the computer do a BLOAD of any program in the disk's directory. Just put the name of the program file before the last zero. This will not cause the program to run, but you can arrange to do that with the machine language code that follows.

6. On top of all the other features you must now write some machine language, even if it's only the RTS to terminate the boot and return to Basic. In this machine language you may do anything you like, including loading files, printing messages, or whatever you think is appropriate.

Some Examples

I have analyzed the contents of three C128 boot disks; what they contain may be interesting for study (and plagiarism) purposes.

First, there's a disk generated by use of the AUTOBOOTMAKER program. Note that the program does not use boot blocks, and (surprisingly) doesn't use the load–a–file feature. It does the whole job in the machine language section, setting up the Basic command, RUN " FILENAME " and then asking Basic to execute this command.

Secondly, there's the CP/M boot disk. Again, no boot blocks, no program load; everything is done in machine language. The program consists of some simple cosmetic setups and then a jump to RAM 0 at address FFD0. The code there will make the switch to CP/M. By the way, the code at FFD0 in RAM did not come from the Kernal . . . technical types might like to puzzle out exactly where it did come from.

Thirdly, the DOS SHELL. No boot blocks. Program DOSSHELL is loaded; the BOOTING message prints a name which is slightly different from the file name. The machine language part moves Basic so that it starts at $5A01 (ugh!); function key F1 is redefined, and an information message is printed.

BANK 12, used by the DOS SHELL program, is not a good bank configuration for you to use. It makes RAM 0 available for addresses up to $7FFF rather than its normal limit of $3FFF. You'd be better off using BANK 15, allowing the machine language program itself to set up more memory space if needed (the program would probably do this with LDA #$0E . . . STA $FF00).

The contents of track 1, sector 0 is annotated in the three examples. With a little study, you too will be able to customize your own boot disks.

You can use a "disk doctor" type of program to write that track and sector, or you can write your own with a U2 type of disk command. Either way . . . have fun.

Disk Created with AUTO BOOTMAKER

The disk was created to load and run a programed called VANCOUVER.

Bytes 00 to 02: " CBM "
Bytes 03 to 06: all zero, no boot blocks.
Bytes 07 to 0F: " VANCOUVER "; name to be displayed in BOOT message
Byte 10 : binary zero (end of display name)
Byte 11 : binary zero (no load name)
Bytes 12 to 18: machine language code:

```
A2 18       LDX #$18
A0 0B       LDY #$0B
4C A5 AF    JMP $AFA5
```

. . .which forces Basic to starting running code starting at $0B19 (and that's the code following, of course).

Bytes 19 to 25: tokenized Basic line, reading:
RUN " VANCOUVER

CP/M Disk

The CP/M track 1, sector 0 starts up this way:

Bytes 00 to 02: " CBM "
Bytes 03 to 06: 00 00 00 00 no boot blocks
Byte 07 : 00, no name for the BOOTING message
Byte 08 : 00, no program name to load
Bytes 09 to 23: machine language code:

```
78          SEI
20 84 FF    JSR $FF84
A9 3E       LDA #$3E
8D 00 FF    STA $FF00 (sort of Bank 0)
A9 C3       LDA #$C3
8D EE FF    STA $FFEE
A9 08       LDA #$08
8D EF FF    STA $FFEF
A9 00       LDA #$00
8D F0 FF    STA $FFF0
4C D0 FF    JMP $FFD0
```

DOS SHELL Disk

The boot block starts up like this:

Bytes 00 to 02: " CBM "
Bytes 03 to 06: 00 00 00 00 no boot blocks
Bytes 07 to 15: name, C128 DOS SHELL, followed by 0
Bytes 16 to 1F: filename, DOS SHELL, followed by 0
Bytes 20 to 73: machine language code:
(Set up key F1):

```
A9 74       LDA #$74
85 2D       STA $2D
A9 0B       LDA #$0B
85 2E       STA $2E
A9 0F       LDA #$0F
85 2F       STA $2F
A9 2D       LDA #$2D
A2 01       LDX #$01
A0 12       LDY #$12
20 21 C0    JSR $C021
```

(A call to FF65 would be more standard)
(Set start–of–Basic to $5A00):
(The three zeros):

```
A0 03       LDY #$03
A9 00       LDA #$00
(0B39) 99 00 5A  STA $5A00,Y
88          DEY
10 FA       BPL 0B39
```

(Basic Start and End pointers):

```
A9 01       LDA #$01
85 2D       STA $2D
A9 5A       LDA #$5A
85 2E       STA $2E
A2 03       LDX #$03
8E 10 12    STX $1210
8D 11 12    STA $1211
```

(Set screen cursor):

```
18          CLC
A2 05       LDX #$05
A0 1D       LDY #$1D
20 F0 FF    JSR $FFF0
```

(Print message):

```
20 7D FF    JSR $FF7D
```

(0B5A) message (DOS SHELL ON, etc.)

```
(0B73) 60   RTS
```

Bytes 74 to 86: Function key definition:
BANK 12:SYS 6656+CHR$(13)

New Loops: The Commodore 128 Basic Stack

Jim Butterfield
Toronto, Ontario

. . .Some programmers have a knack for getting themselves tangled up in code. . .

Even if you're just a Basic person, you've likely heard about the "stack". That's the place where the computer leaves temporary notes for itself. . . how to get back from a subroutine or interrupt, plus other small bits of data.

From the Basic end, the stack holds two items of interest: information on live FOR/NEXT loops, and information on active subroutines. Whenever your program commands FOR. . . or GOSUB. . . the computer notes the command location and the current line number. That's so it will know where to come back to when it encounters a NEXT or RETURN as the case may be, and so it can reinstate the line number in case an error notice is needed.

That's all that is noted for a GOSUB, since all the program needs to know is how to RETURN. The FOR command puts away lots more data, however. It gives the variable identity, the step value, the loop limit, and whether the loop is counting up or down (say, with a STEP –1). That way, NEXT can modify the loop variable by the right amount and then test to see whether to go around the loop again.

All this adds up to: seven items on the stack to log a GOSUB, and eighteen items to record a FOR. These items are reclaimed later – at least they should be. RETURN wipes out the GOSUB entry, and NEXT will kill the FOR entry when the loop has been exercised the proper number of times.

On earlier Commodore machines, this stuff went onto the "hardware" stack. That's the area from 506 going down to 320 (hex $01FA down to $0140), or about 186 locations. That means that you can nest about ten FOR/NEXT loops, or you can go about 24 subroutines deep. Try to go further and you'll get an ?OUTOFMEMORY error, which is puzzling to beginners since there seems to be lots of memory left.

Most programmers get ?OUTOFMEMORY because of sloppy subroutine handling. Within a subroutine, they forget to RETURN and instead leap directly back into the main code with a GOTO statement. The used part of the stack never gets restored, and eventually the program runs out of space. The fastest demonstration of this is the one–line program, 100 GOSUB 100, which will bomb faster than you can say EBCDIC.

The C128

The 128 has a much bigger stack – over 500 bytes – and it's reserved purely for Basic activity. This stack logs not only FOR (eighteen bytes per item) and GOSUB (five bytes per item) but also the new command LOOP (five bytes per item). Even though this stack is much bigger, you can still fill it up quickly with foolish programming.

The new stack is implemented in software, not hardware. It's located in bank 0 in the area from hexadecimal 09FE down to 0800. There's a pointer stored at address 125 and 126 (hex 7D and 7E) which indicates the last location in use. The stack fills from the top down; if the stack is empty, the pointer will say $09FF, if it's nearly full the pointer will have dropped to the low 800's.

Here's a detailed rundown of what goes on the Basic stack for each type of entry:

FOR – hex $81 to indicate a FOR entry;
loop variable address (two bytes);
increment floating value (five bytes);
increment sign, 01 or FF (one byte)
variable limit, floating (five bytes);
line number (two bytes);
return address (two bytes).

GOSUB – hex $8D to indicate a GOSUB entry;
line number (two bytes);
return address (two bytes).

DO – hex $EB to indicate a DO entry;
line number (two bytes);
return address (two bytes).

Mechanics

It's useful to get a feeling for the workings of these. When a Basic program executes a FOR, GOSUB, or DO, the appropriate entry is placed on the Basic stack. Extras: FOR searches to see if the stack contains a previous FOR entry with the same variable name; if so, it strips the stack back to that point and then makes the entry. . . but it can only look through FOR entries. Example: a sequence such as FORJ. . . GOSUB. . . FORJ would work badly; the new FORJ wouldn't catch the previous FORJ that had been started outside the subroutine.

DO may also have an extra: If WHILE or UNTIL is part of this statement, Basic will check and if necessary skip ahead to the appropriate LOOP statement. Note that although a FOR/NEXT must be executed at least once, a DO/LOOP may have its contents skipped entirely – a straight hop from the DO to the LOOP with everything in between ignored.

Now we come to the other end of these constructs. A NEXT will cause all earlier FOR entries to be searched for a matching variable name, although the computer will not search across a GOSUB or DO entry. When the right entry is found, the variable is "stepped" and tested for within range. If it's in range, Basic goes back; if not, the stack entry is scrapped and the program proceeds.

A RETURN will cause a scan of the entire stack. . . the most recent GOSUB entry found will be honoured and the stack stripped back to that point. Note that if FOR loops or DO structures had been opened within the subroutine, they will be scrapped upon RETURN.

A LOOP action depends on whether or not the command is followed by a WHILE or UNTIL. Assuming the LOOP statement is found to be active, it will scan the entire stack. The most recent DO entry found will be honoured and the stack stripped back to that point. Note that if FOR loops or subroutines had been opened within the DO structure, they will be scrapped upon LOOP.

Sample Program

Let's write a C128 program to allow names to be entered, sorted and listed. We'll look at the stack, commenting on some of the workings. Here's the program:

PROGRAM: LOOPER

```
100 DIM N$(100)
110 DO
120 PRINT
130 PRINT " 1 – ENTER NAMES"
140 PRINT " 2 – LIST NAMES"
150 PRINT " 3 – QUIT"
160 PRINT
170 INPUT " YOUR CHOICE";C
180 ON C GOSUB 200,300,400
190 GOTO 170
200 INPUT " NAME";X$
210 J=N
220 DO WHILE J>0
230    K=J–1
240    IF X$>N$(K) THEN EXIT
250    N$(J)=N$(K)
260    J=K
270 LOOP
280 N=N+1
290 N$(J)=X$
300 FOR J=0 TO N–1
310    PRINT N$(J)
320 NEXT J
330 RETURN
400 END
500 LOOP
```

Enter the program; you might like to play with it, entering names and seeing them come out in alphabetic order. In a moment, we'll change it in order to allow ourselves to look around.

You should know that the array N$(..) uses element zero as the first item. Thus, if N (the number of items) equals 3 the array goes from N$(0) to N$(2).

The DO . . . LOOP that extends from lines 220 to 270 is different from the comparable FOR . . . NEXT, and usefully so. Here's why. As we put each item into the table, we compare it with the existing items. But the first time through, there's nothing in the table to compare with. If I coded

```
FOR J=N–1 TO 0 STEP –1
```

. . .I'd be stuck on the first item (where N equals 0), since a FOR/NEXT insists on exercising its contents at least once. No such problem with the DO/LOOP, which skips over the intervening code when the first item is encountered.

We're about to look into the mechanics a little. If you don't like tinkering with the works, or if you feel threatened by hexadecimal numbers, you might prefer to skip this section.

We want to analyze the stack, just to show that it's there and can be viewed whenever you like. Let's put a STOP command in at line 305:

```
305 STOP
```

RUN the program once again; call option 1 and enter any name. When the program stops, command MONITOR.

Now we'll examine the Basic stack pointer. Command M7D7E and look at the line that results. The first two bytes should be E3 09, meaning that the stack pointer is down to 09E3. Good – we'll look at the stack with command M9E39FF.

Let's review what we know of the program's "state". The first thing that happened is that line 110 executed a DO. When we selected option 1, line 180 performed a GOSUB. Line 220 performed another DO, but the LOOP at line 270 cancelled it. The next thing that happened, just before the STOP at 305, was that a FOR loop was opened by line 300.

So we have a FOR entry inside a GOSUB entry inside a DO entry. Let's see if we can find them on the stack.

Address 9E3 contains a value of 81 – that's the FOR flag. You can see that the variable is at address 0410 (variables are kept in Bank 1). Floating point numbers are hard to read, even if you know the secret, but beyond them we can see 2C 01 for line number 012C – that's hex for 300.

Eighteen locations along, at 9F5, we see the 8D flag signalling a GOSUB entry. Again, we could read the line number entry as hex 00B4 or 180. Five more locations along, at 9FA, we see EB for the LOOP item, with line number 6E for line 110.

It's all there, and you can look at it any time you get fuddled over loops. Now return to Basic with command X, and then delete line 305.

The Untangler

Some programmers have a knack for getting themselves tangled up in code. I get calls that sound like this: "I'm seven subroutines deep, and now I've found a situation where I want to give up and return to the menu. How do I get out?"

The proper answer is: you should never have gotten yourself into that mess. Start over, use variable flags to give return signals, and next time do it right.

In the past, I've taken pity on some of these unfortunates by digging out a SYS or suggesting a brief machine language routine to set things to rights. At least these people have learned (sometimes the hard way) that you can't just go back to the menu with a GOTO or you'll leave a messy stack and eventually get an OUTOFMEMORY message.

But with the 128, there's an easier way out. If you have only one DO/LOOP active, you can clean out all subroutine and FOR/NEXT entries just by going to the LOOP.

Examine the sample program. Note that a DO appears very early in the coding, and that the corresponding LOOP can never be reached. Here's the trick: if we ever do get to that LOOP – assuming no other DO's are open – we'll immediately be transported back to the menu and the stack will be neatly trimmed. It's a little like the magic word FROBOZZ that transports you back to the vault.

Here's a simple example. Suppose in the above program, we've picked option 1 and are asked to enter a name. At this point, we say something like, "Gosh . . . I don't really want to enter a name after all". It would be nice to abort back to the menu.

Suppose we say, "OK, if the user types an asterisk character instead of a name we'll go back to the menu". We might try a new line at 205 which says something like:

```
IF X$ = "*" GOTO 120
```

But if we do so, we'll eventually have problems, since we have never cleared away the subroutine call from line 180.

The solution is simple. Enter line 205 as:

```
205 IF X$ = "*" GOTO 500
```

When an asterisk is entered, we'll go to the LOOP statement at line 500. This will search for the last corresponding DO, finding it at line 110. The Basic stack will be trimmed back to that point, in this case removing the subroutine entry, and the program will resume by printing the menu.

It's a simple example, and doesn't trim much from the stack. But once you understand the principle, you can use it more generally.

Eliminating SAVE@ And Other 1541 Bugs

Phillip A. Slaymaker
Palm Beach Gardens, FL
Copyright 1986, P.A.S

At Long Last! – The Code To Do It!

The 1541 Disk Operating System (DOS) has a number of bugs or idiosyncrasies, the Save–with–Replace command (SAVE@) bug in particular, which have plagued Commodore users for a number of years. This article presents a number of patches to the 1541 DOS which fix the SAVE@ bug and a few other related bugs under most conditions. These changes have been programmed into EPROMS and tested successfully in two 1541 drives.

The SAVE@ bug has been a continuing subject of controversy in numerous issues of The Transactor. The author recently published a two part explanation of SAVE@ in COMPUTE! October and November 1985, "Save With Replace: Debugged At Last" including a program which unequivocally demonstrated the bug's existence. A review of the reasons for the bug will be given – for a full review the reader should refer to the aforementioned articles.

As part of our testing of 1541 DOS for the SAVE@ bug, we have reviewed the 1541 source code. It is documented in two good sources. A complete disassembly of one version of DOS V2.6 is presented with comments in "The Anatomy of the 1541", by L. Englisch, et al, 1984 from Abacus Software. A full analysis of the DOS V2.6 routines, specified by name and address, is given in "Inside Commodore DOS", by R. Immers and G. Neufeld, 1984 from Datamost. We will use the DOS subroutine names and addresses listed in "Inside Commodore DOS" since most people who will read this will have a copy of the book.

Our testing was done by creating a master disk with three directory sectors of files. All test sequences were done starting with a fresh copy of the master disk and with the drive in a completely reset state. The disk drive memory and the disk BAM, directory, and sector data were examined after each SAVE@ command using our Peek A Byte 64 disk and memory utility (available from Quantum Software, P.O. Box 12716, Lake Park, FL 33403, 305–840–0249). Internal pointers were noted before and after each load or SAVE@ with the drive number specified only in the SAVE@. These tests were repeated with the drive number always specified. Tests were also performed using new EPROMS which were burned for the 1541. This was done to test possible patches to the DOS and to help diagnose the drive operation.

SAVE@ Bug Explanation

DOS V2.6 has 5 internal buffers, with buffers 0 to 4 starting at memory pages $300, $400, $500, $600, and $700, respectively. DOS assigns channels and buffers to the BAM, directory sector, and file sectors being read or written. Normally DOS assigns two read or two write channels and uses only 3 of the 5 buffers. The SAVE@ command, however, requires all 5 buffers – two read, two write, and the BAM. If the DOS can't find a free buffer, then the DOS tries to steal an assigned, but inactive buffer. If the BAM is stolen, SAVE@ may fail.

Not specifying the drive number can cause a buffer to be stolen. When the directory is accessed, AUTOI ($C63D) reads the BAM of the disk in the specified drive, and also tries to initialize drive 1 if no drive was specified. Usually buffer 3 ($600) is allocated for the phantom drive 1 (not present in a 1541 drive) BAM and a B1 SEEK command is issued to the disk controller. An internal DRIVE NOT READY error occurs in the disk controller and puts the error code #$0F in the job queue at $03. This error code is then trapped by AUTOI and not reported outside the disk drive. This leaves buffer 3 allocated but inactive.

Since the SAVE@ command requires all 5 buffers and only 4 are now available, DOS steals an inactive buffer by calling STLBUF ($D339). STLBUF can be called several times during a SAVE@ command causing the BAM and directory sectors to be reassigned to different buffers during a single SAVE@. STLBUF ($D339) should not steal the drive 0 BAM, but should instead steal back the unused buffer incorrectly assigned to drive 1. It never steals the drive 1 BAM buffer #3 at $600 because STLBUF cannot take a buffer if the buffer had a drive error occur.

SAVE@ works most of the time because after the BAM is stolen, it is read back in when needed and updated using the BAM images. A BAM image for each of two tracks is stored at BAM ($2A1 – $2B0). Each time a new block is allocated by WUSED ($EF90), it is in the BAM image. When a new track is tested for free sectors, the DOS checks if it has a BAM image for it. If not, it calls SWAP ($F05B) which first updates the BAM with the BAM image from the second to last track, copies the new track's BAM map into the BAM image, and then zeros that track in the BAM. Only TWO tracks can be updated, however, since there are only two images. If more than two tracks have been accessed by SAVE@, the BAM may NOT be correctly updated. A track may either be updated correctly, be left unchanged or may be fully allocated, depending on when the BAM was stolen.

Possible DOS ROM Modifications

We have tried several modifications to the DOS by programing EPROMS. These included:

1) STLBUF ($D339) should be modified to allow stealing a buffer with a DRIVE NOT READY error. It could also be prevented from stealing the drive 0 BAM buffer, but this may be required and so this second patch was not done.

2) A drive 0 could be forced by modifying ONEDRV ($C312) and SETDRV ($C33C) to set drive 0 rather than setting the default drive. Alternately, the DOS could be prevented from ever switching to drive 1. We tried modifying TOGDRV ($C38F) so only drive 0 is allowed and this patch works. Other routines must be modified to keep from searching the directory twice. However, these DOS

patches can't be done on a dual drive and so were not implemented.

3) More drive memory and buffers should be added. We added 8K of memory to our drive, but a hardware change is beyond most individuals.

Recommended Modifications

We made a number of modifications which correct a number of bugs in the 1541 V2.6 DOS release 5. (The release number of the $E000 ROM is 901229-05. The $C000 ROM was never changed and is 325302-01). The addresses shown assume that the ROMs were read into the computer memory from $2000 - $5FFF (ROM addresses $C000 - $DFFF and $E000 - $FFFF) in preparation for burning new EPROMs. All of the extensive patch code starts at $C010 for convenience.

The first patch replaces the incorrect use of zero page indexed addressing with absolute addressing for the NODRV ($FF and $100) flags which indicate whether a drive is present. Since $FF,X addressing "wraps around" to $00, the flag for no drive 1 was stored in the job queue and caused the 74, DRIVE NOT READY error (which occurred all too often). It only indirectly affected the SAVE@ bug.

```
2010 a6 7f       ldx  $7f      DRVNUM normally 0
2012 bd ff 00    lda  $00ff,x  get NODRV flag
2015 60          rts
2016 98          tya
2017 9d ff 00    sta  $00ff,x  store NODRV flag
201a 60          rts
201b a9 00       lda  #$00
201d f0 f8       beq  $2017
201f a9 00       lda  #$00     set WPSW write
2021 95 1c       sta  $1c,x    protect status to 0
2023 f0 f2       beq  $2017
```

The next patch corrects a bug which put the value #$02 at $197 in the drive memory. This bug can cause problems with programs which download routines into the disk drive (especially fast loading programs). The calling routine prevents y from exceeding #$0C - a value of y = #$FE causes $197 to be addressed.

```
2025 0a          asl          called from $DCBE and $DCCB
2026 a8          tay          times 2
2027 a9 02       lda  #$02
2029 99 99 00    sta  $0099,y
202c 60          rts
```

STLBUF ($D339) must be allowed to steal a buffer with an #$0F error (DRIVE NOT READY) - the following patch does this.

```
202d c9 0f       cmp #$0f     compare error code with #$0f
202f d0 03       bne $2034
2031 4c 73 d3    jmp $d373    steal if equal
2034 a6 6f       ldx $6f      else
2036 e0 07       cpx #$07
2038 4c 6f d3    jmp $d36f    check next channel
```

A possible SAVE@ related bug is in the serial bus communication routines. Occasionally the ATN (attention) interrupt flag bit of $180D INTERRUPT FLAG REGISTER is not cleared by the TSTATN routine ($EA59). Addressing $180F (or $1801) clears the flag bit.

Gerry Neufeld's theory of a serial bus ATN related SAVE@ bug is presented in INFO issue #9 Dec 85 - Jan 86 by the magazine's editors. Briefly, it is claimed that the drive listens to the NMI (non-maskable interrupt) line from the computer before the drive is done updating the BAM and directory. If a new command is sent before SAVE@ is completed, the BAM or directory may not be updated correctly since the new command will be executed. The article incorrectly refers to ATN signal as an NMI line - the ATN signal can be interrupt disabled - and the technical details or proof of the theory are not given. I have not tested his theory either, however, the ATN bug patch given below should prevent the drive from prematurely listening to the serial bus from the computer.

```
203b ad 0f 18    lda  $180f    clear ATN interrupt flag
203e 4c 5b e8    jmp  $e85b
2041 ad 0f 18    lda  $180f    clear ATN interrupt flag
2044 4c d7 e8    jmp  $e8d7
```

The following routine patches the interrupt handling routine so that a user supplied routine stored in the drive RAM will intercept every interrupt, whether caused by the serial bus or by the internal timer. As written it requires that extra RAM be installed in the drive at $A000 - $BFFF. (A memory expansion kit and instructions are available from CSM Software, Inc., P.O. Box 563, Crown Point, IN 46307, 219-663-4335). The routine can be used to investigate interrupt driven disk routines including the SAVE@ bug.

```
2047 ad fe ff    lda  $fffe    #$67  pattern to match
204a ac ff ff    ldy  $ffff    #$ff      "
204d cd 00 a0    cmp  $a000    must be #$67
2050 d0 08       bne  $205a
2052 cc 01 a0    cpy  $a001    must be #$ff
2055 d0 03       bne  $205a    skip routine if pattern wrong
2057 20 5e c0    jsr  $c05e    jump to subroutine
205a ad 0d 18    lda  $180d
205d 60          rts
205e 6c 02 a0    jmp  ($a002)  specified in $A002 and $A003
```

The following patches are short and are explained in the comments.

```
21b3 ea          nop
21b4 20 1b c0    jsr  $c01b    store #$00 in NODRV

2661 20 16 c0    jsr  $c016    store .A in NODRV
2664 d0 03       bne  $2669
2666 20 42 d0    jsr  $d042
2669 a6 7f       ldx  $7f
266b 4c 12 c0    jmp  $c012    load .A with NODRV

3021 20 45 e6    jsr  $e645    call CMDER2 instead

3071 ea          nop
3072 20 1f c0    jsr  $c01f    initialize WPSW and NODRV

336b 4c 2d c0    jmp  $c02d    check for #$0F error code
336e ea          nop

3cba c9 07       cmp #$07      be sure .A < #$07
3cbc b0 03       bcs  $3cc1
3cbe 20 25 c0    jsr  $c025    store #$02 in $0099,Y
3cc1 b5 ae       lda  $ae,x
3cc3 09 80       ora  #$80
3cc5 95 ae       sta  $ae,x
3cc7 c9 07       cmp #$07      be sure .A < #$07
3cc9 b0 03       bcs  $3cce
3ccb 20 25 c0    jsr  $c025    store #$02 in $0099,Y

.:45b7 d0 41 53 20 44 4f 53 20    pas dos - author's ID
```

```
4a68  4c  3b  c0    jmp  $c03b   TSTATN patches
4a6b  4c  41  c0    jmp  $c041

4ae4  ea          nop           eliminate ROM
4ae5  ea          nop           checksum routine
4ae6  e0  c0      cpx  #$c0
4ae8  ea          nop
4ae9  ea          nop

4c04  4c  3b  c0    jmp  $c03b   TSTATN patch

5017  ea          nop
5018  20  10  c0    jsr  $c010   get NODRV
501b  f0  05        beq  $5022
501d  a9  74        lda  #$74
501f  20  45  e6    jsr  $e645   call CMDER2 instead

5e6c  20  47  c0    jsr  $c047   interrupt routine patch
```

The following list summarizes the memory locations of all the changes.

```
2010 2011 2012 2013 2014 2015 2016 2017 2018 2019
201a 201b 201c 201d 201e 201f 2020 2021 2022 2023
2024 2025 2026 2027 2028 2029 202a 202b 202c 202d
202e 202f 2030 2031 2032 2033 2034 2035 2036 2037
2038 2039 203a 203b 203c 203d 203e 203f 2040 2041
2042 2043 2044 2045 2046 2047 2048 2049 204a 204b
204c 204d 204e 204f 2050 2051 2052 2053 2054 2055
2056 2057 2058 2059 205a 205b 205c 205d 205e 205f
2060 21b3 21b4 21b5 21b6 2661 2662 2663 266b 266c
266d 3022 3071 3072 3073 3074 336b 336c 336d 336e
3cba 3cbb 3cbc 3cbd 3cbe 3cbf 3cc0 3cc7 3cc8 3cc9
3cca 3ccb 3ccc 3ccd 45b7 45b8 45b9 4a69 4a6a 4a6c
4a6d 4ae4 4ae5 4ae8 4ae9 4c05 4c06 5017 5018 5019
501a 5020 5e6c 5e6d 5e6e
```

Burning New EPROMs

A full discussion of burning EPROMs is beyond the scope of this article. Two good sources of information and EPROM hardware are CSM Software, Inc. and their EPROM Programmers Handbook, or Jason–Ranheim Company (580 Parrott Street, San Jose, CA, 95112). The 1541 drive ROMs are not always in sockets. Obtain professional help if desoldering of the ROMs is required – it is not for beginners. Two 2764 ROMs and two 28 pin to 24 pin adapters will be required. If the ROMs are easily removable, they should be read with the EPROM programmer into memory starting at $2000 for the $C000 ROM and at $4000 for the $E000 ROM so that they correspond to the addresses given in this article. The patches may be made with any machine language monitor program, either in mini–assembler or in HEX.

Editor's Notes

Well it's about time I got to write the following: Thank you Mr. Phil Slaymaker, not only for the article, but for taking the time to eliminate the longest running, unsolved bug mystery. Your laborious research will be appreciated world-wide, and so it should. I hereby nominate Phillip A. Slaymaker for a spot in the Commodore Hall of Fame. Anyone second it?

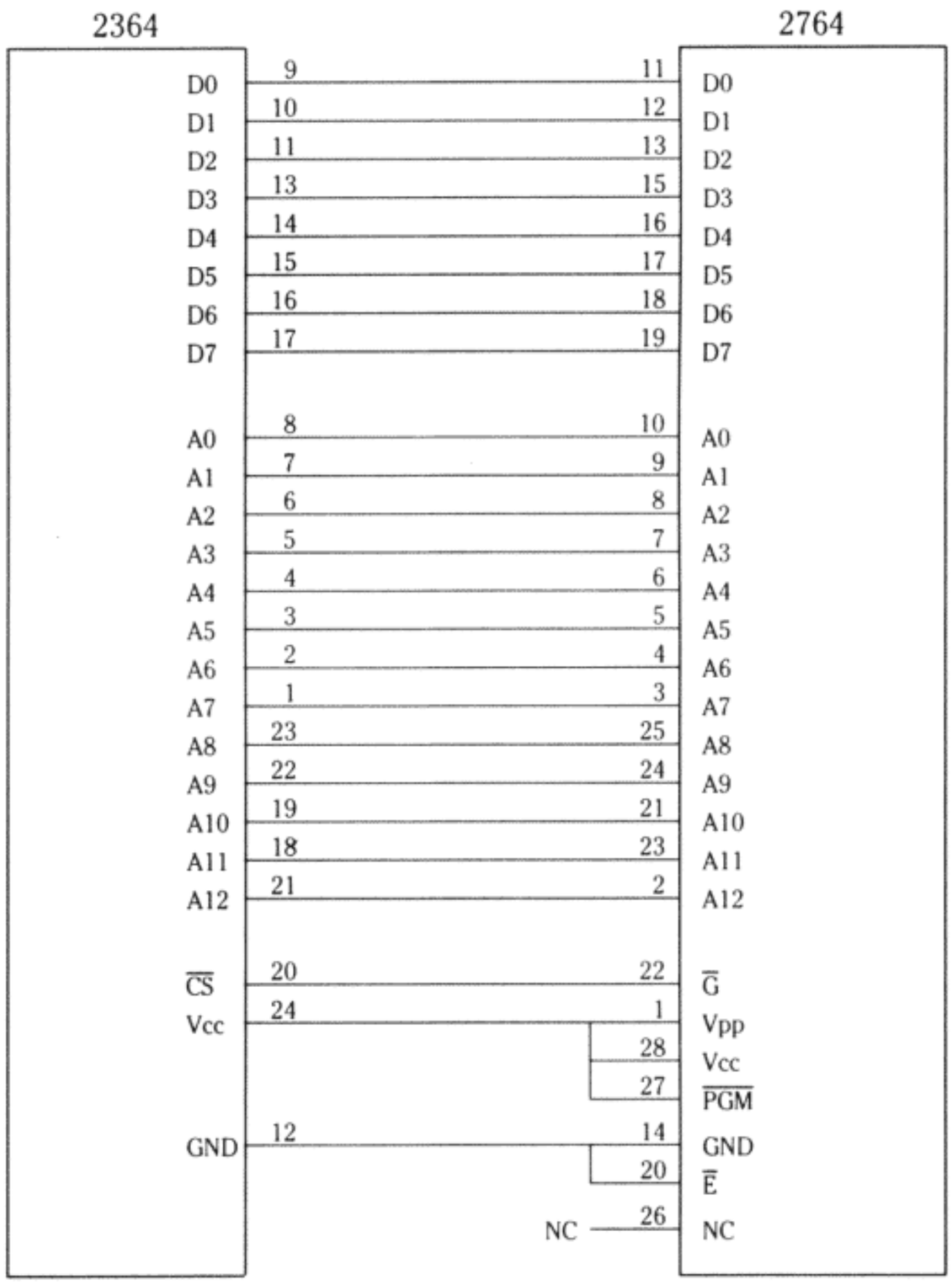
However. . . , there's only one slight problem. I still contend that the SAVE@ bug was first reported before the 1541 was built. The 4040 was the first carrier of this plague, but since these beasts are fast becoming dinosaurs, and the fact that we can't seem to force out a demonstration of the hap, I must officially surrender this battle until such time that both necessity and solid evidence can be supplied to once again open the file. Therefore, as they say, "the case rests".

For those who wish to burn chips with these patches, there are two approaches. The first, as suggested in the article, is with the popular 2764 EPROMs. The 28 to 24 pin adapters are also available from CSM, but for the so inclined we've supplied assembly instructions below. Unfortunately, with the adpater in place, chances are the top of the 1541 case will not fit back on. Although this "blessing in disguise" keeps the drive cooler, certain situations may warrant. . .

The second, if money is no object, is with Motorola's MCM68764 EPROMs which are pin-for-pin compatible with the Commodore ROMs. You'll need two, and they're about $16.00 a piece (available from Jameco among others).

Commodore is well aware of Mr. Slaymaker's findings, and word is that new 1571s will soon be equipped with Revision 4 ROMs that fix these bugs (currently all 1571s house Revision 3 ROMs). An update kit for the 1541 is NOT currently under consideration, but we'll keep you posted if we hear otherwise.

2764 (28 pin) TO 2364 (24 pin) Adapter



Use a 28 pin WW socket and a 24 pin ribbon cable header (male). On the 28 pin dip, cut down pins 1, 2, 20, 23, 26, 27, and 28 to about 3/8". Using wire wrap, short pins 1, 27, and 28. Connect pin 27 to pin 24 of the cable header, pin 23 to pin 18, pin 20 to pin 12, and pin 2 to pin 21. The long pins will plug directly into the cable header such that pin 3 of the 28 pin WW goes to pin 1 of the 24 pin header (pin 4 to pin 2, 5 to 3, etc, 25 to 23, 26 to 24).

FORMAT TRACK 36

David A. Hook
Barrie, Ontario
© 1986 David A. Hook

. . .the appeal of such a program? In my case,
I wished to perform an elementary disk protection scheme. . .

Michael Mossman's article (1541 RAM Expander) in Transactor Vol. 6, Issue 5 really caught my fancy. He described how to add a 3K VIC RAM cartridge to the 1541 drive, thereby expanding that device's capability to store programs for direct execution within the disk drive.

He also provided a couple of example programs: one demonstrated how to format extra tracks on a diskette, beyond the normal 35 provided by Commodore's DOS (which is stored in ROM, of course). The extra RAM allowed him to copy the disk ROM code, substituting his modifications. By executing this code directly (from its new RAM location), the new format routine could generate these extra tracks. (Mr. Mossman indicated 38 tracks would be formatted, however only 37 are actually done). Nonetheless, the idea is fascinating.

With my acknowledged clumsiness with a soldering iron, and an expensive disk drive at stake, I sought an alternate to the construction project. Armed with the two 1541 books I reviewed previously (Ref. 1, 2), the task was completed successfully.

I used the normal disk formatting routine to set things up. Then a machine language program goes into disk RAM buffer #3. Perform a seek operation to Track 35, then execute the program from disk RAM. Much of the code used by Mr. Mossman comes from the 1541 ROM--752 bytes of the 3K extra RAM was consumed. In my case, I needed to do everything in the minimum space of an unmodified 1541--preferably less than 256 bytes (or one disk buffer's worth). The Immers/Neufeld book was invaluable, and my program borrows from a couple of their creations.

Be very careful with the values in the DATA statements. By programming the Floppy Disk Controller (FDC) directly, you are bypassing all the checks and safeguards that Commodore put in there. If you tell the drive head to go to Track100, it will happily try to oblige. Immers & Neufeld speak with experience in describing the need to disassemble the drive to effect adjustments when this happens. Needless to say, neither I nor the Transactor can accept any liability for damage wrought by use of these routines, but they have been checked on two different drives.

Well what was the appeal of such a program? In my case, I wished to perform an elementary disk protection scheme, without bumping the drive head. By formatting a disk (with 36 tracks) using one ID sequence then reformatting (normally) with a different ID, I had its foundation. The main program to be protected included a seek routine to Track36 which picked up the special ID and used it as a key variable in the program. Since the main program was compiled, this was not very visible. When the program could not get the special ID, it caused wrong answers to be generated. As the calculations were fairly complex, these wrong answers were not obvious--a nasty trick to use in a financial type program!

For machine language fans, the PAL source code has been provided.

References:

1. Inside Commodore DOS by Richard Immers and Gerald Neufeld. Datamost published this originally (1983). Now published by Brady Communications Company, c/o Prentice Hall, W. Nyack, NY 10994.

2. The Anatomy of the 1541 Disk Drive by Lothar Englisch and Norbert Szczepanowski. Abacus Software, Grand Rapids, MI 49510.

Format > 35: BASIC Loader

```
MK  10 rem format a track > 35
LF  20 rem (c) 1986 d. a. hook, all rights reserved
PK  30 print " Sinsert blank disk, then
       press <return> "
KM  40 get z$: if z$<>chr$(13) goto 40
ID  50 input " qq disk name, id "; n$, id$
GO  60 open 15, 8, 15, " n0: " + n$ + " , " + id$:
       gosub 800
```

```
JN | 70 count=0
PB | 99 rem put jmp $0400 into buffer #3
KO | 100 print#15, "m-w"; chr$(0); chr$(6); chr$(3);
   |     chr$(76); chr$(0); chr$(4)
AC | 110 gosub 800
PB | 199 rem set to track 35/sector 0
HH | 200 print#15, "m-w"; chr$(12); chr$(0); chr$(2);
   |     chr$(35); chr$(0)
EI | 210 gosub 800
IA | 299 rem read ml into disk ram ($0400)
JH | 300 restore: for i=0 to 69: read d
HO | 310 print#15, "m-w"; chr$(i); chr$(4); chr$(1);
   |     chr$(d)
EE | 320 next
EH | 399 rem execute from $0003
JE | 400 print#15, "m-w"; chr$(3); chr$(0); chr$(1);
   |     chr$(224)
MG | 499 rem read disk error code
JD | 500 print#15, "m-r"; chr$(3); chr$(0)
NA | 510 get#15, e$: e = asc(e$)
FL | 520 if e=1 then print "done, ok": close 15: end
CI | 530 if count=10 goto 900
NC | 540 count=count+1: goto 100
AM | 800 input#15, e, e$, t, s: if e=0 then return
GL | 810 print e; e$; t; s; "failed": close 15: end
GE | 900 if e<17 then print "controller error #"; e
AM | 910 gosub 800: goto 810
PO | 1000 data 165,  34, 133,  81, 169,   1, 141,  32
OH | 1010 data   6, 169,  64, 141,  33,   6, 169,  15
FE | 1020 data 141,  34,   6, 169,  16, 133,  67,  32
CD | 1030 data  40,   4,  32,  40,   4, 230,  81, 165
PB | 1040 data  81, 201,  36, 144, 242,  76,  12, 251
LK | 1041 rem                ↑↑ highest track number
   |          (>35) to be formatted
BC | 1050 data 174,   0,  28, 232, 138,  41,   3, 133
AF | 1060 data  20, 173,   0,  28,  41, 252,   5,  20
LB | 1070 data 141,   0,  28, 160,   5, 162, 255, 202
BG | 1080 data 208, 253, 136, 208, 250,  96
```

Format > 35: PAL Source Code

```
NP | 100 rem format track 36 source code
EK | 110 rem (c) 1986 d. a. hook
GJ | 120 rem
IN | 130 rem code stored in disk buffer#3
OL | 140 rem loaded from the basic program
EL | 150 rem
EA | 160 sys 700 ;for pal 64 assembler
AC | 170 ;
KJ | 180 *=$0400
ED | 190 ;
PN | 200 ;*** 1541 ram locations ***
IE | 210 ;
OE | 220 drvtrk  =   $22    ;current track number
HI | 230 sectr   =   $43    ;sector counter
FE | 240 ftnum   =   $51    ;format track number
AH | 250 ;
JC | 260 dskcnt  =   $1c00  ;disk controller
EI | 270 ;
OI | 280 ;
GN | 290 start  lda   drvtrk  ;current track #
GM | 300        sta   ftnum   ;keep it
MK | 310 ;
NL | 320        lda   #1      ;no errors allowed
OA | 330        sta   $0620
KM | 340 ;
MI | 350        lda   #$40    ;trial # of bytes on 1/2
   |                          track
NC | 360        sta   $0621
NO | 370        lda   #$0f
CE | 380        sta   $0622
MP | 390 ;
DE | 400        lda   #16     ;allow 16 sectors/track
II | 410        sta   sectr
KB | 420 ;
KM | 430 bigger jsr   step   ;move head twice
GD | 440        jsr   step
DM | 450        inc   ftnum
NK | 460        lda   ftnum
CN | 470        cmp   #36     ;are we there yet
HO | 480        bcc   bigger
AG | 490 ;
GB | 500        jmp   $fb0c   ;enter rom routine now
EH | 510 ;
GH | 520 step   ldx   dskcnt  ;move head out 1/2 track
OO | 530        inx
AN | 540        txa
EB | 550        and   #%11
II | 560        sta   $14     ;unused ram loc'n
PG | 570        lda   dskcnt
MG | 580        and   #%11111100
KJ | 590        ora   $14
LM | 600        sta   dskcnt
IN | 610 ;
EO | 620        ldy   #5      ;delay loop (for head
   |                          settling)
PI | 630        ldx   #$ff
GP | 640 ;
CF | 650        loop  dex
NC | 660        bne   loop
JF | 670        dey
BE | 680        bne   loop
OJ | 690        rts
CD | 700 ;
CK | 710 .end
```

An Amiga Parallel Printer Cable

Steve Michel
Sterling, IL

In the Volume 6, Issue 6, the Article, "The Amiga: A User's Perspective" touted the Amiga'a ability to work easily with any printer. It didn't mention, however, the difficulty involved in hooking up a printer in the first place. This article addresses that problem.

If you have tried to hook your brand new AMIGA to a Centronics–type parallel interface printer, you probably ran into a little trouble. Well, you aren't alone. So did everyone else in the world!!

The reason for this problem is fairly simple. The 25 pin connector at the rear of the AMIGA that is designated as the parallel port is a called a DB–25 connector. Your printer cable has a DB–25 connector on one end to attach to the AMIGA and a 36 pin connector on the other end to attach to your printer. These connectors are fairly standard connectors throughout the industry and have allowed us the limited amount of compatibility between computers and peripherals that we now enjoy.

So, if the AMIGA and the printer cable use the same standard connectors, what's the problem ? All connectors come in two forms: MALE and FEMALE. In order to connect two cables together, one must have a female connector and the other must have a male connector. The standard printer cable that you would purchase has a male DB–25 connector. Guess what type of DB–25 connector is supplied in the rear of the AMIGA? If you said male, you must be a long-time Commodore user and used to such goings on.

The use of a male DB–25 connector on the AMIGA puzzled me at first, especially since two other connectors in the rear (DISK DRIVE and SERIAL) are female connectors. However, after a little checking in my printer manual and the Introduction to AMIGA manual, I found that some pins on the AMIGA DB–25 are not directly connectible with a standard off–the–shelf printer cable. Therefore, in an attempt to keep an unwary user from just plugging in a cable that could cause some serious damage to either the AMIGA or the printer, a male connector was used on the AMIGA. This forces the user to obtain the correct cable.

There are three possible solutions to the above problem: (1) don't use a printer, (2) wait for Commodore or some other second source to produce an appropriate cable or (3) construct your own cable.

Option 1 is unacceptable. Option 2 is okay if you are the patient type. The remainder of this article focuses on Option 3.

Table 1 shows the pin connections necessary to correctly interface the AMIGA parallel DB–25 connector with a 36–pin parallel connector for a Centronics interface printer. The parallel function labels and pins numbers were obtained from a Panasonic KX–P1090 printer manual. The AMIGA function labels and pin numbers were obtained from the Introduction to AMIGA manual (Reference 7–13). The wire colors column has been left blank to assist in creating your own cable.

TABLE 1

Parallel Function Label	Wire Colour (fill in)	Pin Number Parallel Connector	Pin Number Female DB–25 Connector	Amiga Function Label
STROBE		1	1	DRDY
DATA1		2	2	D0
DATA2		3	3	D1
DATA3		4	4	D2
DATA4		5	5	D3
DATA5		6	6	D4
DATA6		7	7	D5
DATA7		8	8	D6
DATA8		9	9	D7
ACK		10	10	ACK
BUSY		11	11	BUSY
PE		12	12	POUT
SELECT		13	13	SEL
GND		19	14	GND
GND		20	15	GND
GND		21	16	GND
GND		22	17	GND
GND		23	18	GND
GND		24	19	GND
GND		25	20	GND
GND		26	21	GND
GND		27	22	GND
PRIME		31	25	RESET

Making The Cable:

1. Obtain the necessary cable components: (the parts numbers are Radio Shack numbers)

36 pin male Centronics connector (276–1534A)
25 pin female DB–25 connector (276–1548)
Headshells for both connectors
An appropriate length of 25 conductor cable

If you have access to a good electronics store, the above list should prove no problem. However, in my area the 25 conductor cable was impossible to obtain. The next best approach was to buy a standard printer cable with connectors already in place and alter it to fit the AMIGA. In this case, you only need the standard cable (26–1401) and the 25 pin female DB–25 connector. As an alternative, 25–conductor ribbon cable will work well.

2. If altering a standard printer cable, go to STEP 4.

3. Each wire in the 25 conductor cable will be color coded. Choose any color for pin 1 and write it in Table 1. Continue selecting a different coloured wire for each of the 23 pins listed in the Table. The two excess wires may be snipped off. Go to STEP 6.

4. Remove the hood from the 25 pin male DB–25 connector and expose the wires at the rear of the connector. Remove the connector completely by snipping all the wires. Save this male connector for future projects.

5. Using a continuity tester or Ohm meter, decipher the color coding for each pin on the 36 pin parallel connector. Do this by placing one lead of the continuity tester on pin 1 of the parallel connector and then systematically testing each of the wire ends that were just disconnected from the male DB–25 connector. When the continuity tester light (or buzzer) turns on, write that wire color in Table1. Continue this process for each of 23 pins listed. Be sure to use the pin numbers listed under the parallel connector column. When this has been completed, there will probably be several wires "left over". That is because only 23 of the possible 36 are going to be used. These "left over" wires may be simply snipped off making sure that no exposed metal ends come in contact with each other.

6. Using Table 1, solder each of the colour coded wires to the correct pin numbers on the DB–25 connector. (If starting with all separate components, solder the same coloured wire to the corresponding pin on the parallel connector as well.) Please note that the first thirteen wires are connected "straight through" and the first pin number change occurs at pins 19 on the parallel and 14 on the DB–25.

7. When all the wires have been soldered, RE–CHECK each of the pin connections with a continuity tester one last time to insure that all is right with the world. Too much haste at this point could be deadly to your AMIGA and/or printer.

8. Replace the headshell(s) on the connector(s).

9. Connect the cable and you should be on your way. Happy printing.

TransBloopers

Volume 7, Issue 01, page 54

The sub-heading for the article "Adding Functions to Basic" should not be there. It says "Execute machine language programs inside your 1541", which came from an article in a previous issue and has absolutely nothing to with this one. Also, the source listing in the same article (on pages 56 and 57) was converted to PAL assembler format from CBM format, and all of the question marks in the comments were tokenized to read as PRINT. A clear case of the 3AM-make-it-for-deadline disease, which attacks us mercilessly from time to time.

Volume 7, Issue 01, page 30

The unexpected happened in our last issue. Jim Butterfield's C-128 RAM map was not quite perfect! On page 30 you will find three mistakes. All of them are in Bank Zero. First, the correct decimal equivalent of $0A1D-$0A1F is 2589-2591. Second, the correct decimal equivalent of $1C00-$1FF7 is 7168-8183. And, finally, the correct decimal equivalent of $1FF8-$1FFF is 8184-8191. We hope that these minor errors haven't fouled up your programming efforts too much.

Transactor Disk #12 Fix

We've been receiving reports of some real problems with Disk #12. Somehow the file pointers of the first 7 entries each point to the *next* entry in the directory. That is, if you try loading the second program in the dir, you'll actually get the third program, loading the 3rd gives you the 4th, etc. The 9th file and all afterwards are ok. The following program will fix this problem, but don't run it with any other disk in the drive but Transactor Disk #12. As written it should work on 1541/40, 4040, MSD, Indus, or any other 35 track compatible.

```
HF  100 rem save " 0:the fix " ,8
DH  110 rem ** rte/86 – to fix transactor's disk #12
FK  120 rem ** when complete, all files but 'aid4' and
PH  130 rem ** 'vic aid.rel' will have been repaired
FE  140 dv = 8: open 15,dv,15: open 8,dv,8, " # " :
        print#15, " u1: " 8;0;18;1
DB  150 for x = 0 to 7: for y = 0 to 31
PB  160 get#8,a$: a$(x) = a$(x) + chr$(asc(a$ + chr$(0))):
        next y,x
CB  170 if mid$(a$(0),4,1) = chr$(0) then 230: rem has
        already been repaired
JA  180 rp$ = chr$(0) + chr$(0): for x = 0 to 7
DP  190 t$ = mid$(a$(x),4,2): a$(x) = mid$(a$(x),1,3) +
        rp$ + mid$(a$(x),6): rp$ = t$
IP  200 next: print#15, " b–p: " 8;0
OI  210 for x = 0 to 7: print#8,a$(x);: next
CH  220 print#15, " u2: " 8;0;18;1
PL  230 close8: close15: end
```

We noticed some other problems too. Some stray glitch really gave us the slip! Besides "AID4" and "VIC AID.REL" (as mentioned in the rems above), "VERIFIZER.PET" is also virtually a write-off, but these are on most of the other Transactor Disks should you need them. "C128 MESSAGE" and "ARCHITESTER" for the C128, both load ok and look ok, but are not tokenized properly for correct operation on the C128. Both are short. . . simply load them and hit RETURN over each line. SAVE them back to disk and they'll be fine forever after.

Save SYMASS Symbols

Robert Huehn
Neustadt, Ontario

Create Symbol Table Files Accessible From The BASIC Editor

SYMASS 3.1 Assembler Notes

SYMASS 3.1 has a few features not mentioned in last issue's article.

1. Binary numbers can be used with the % prefix.

2. The offending line is listed when an error condition occurs.

3. All instructions like 'lda 0,y', which have no zero page mode, are assumed to be absolute.

SYMASS 3.1 has two bugs, one which prevents using the opcode TXS. At the beginning of the source code NOPS should equal 56, not 55, to let the last op in the table be recognized. Also, the high byte of an addition isn't stored to memory, only the low one, so values greater than 255 won't work. The fix is easy enough:

Load in SYMASS 3.1, and type:

```
poke 3304, 56
poke 2965, 234
poke 5057, 50
```

Save the updated version as SYMASS 3.12.

All programmers should be aware that 'def' is not only a BASIC token, but also a valid hexadecimal number. SYMASS will give an illegal quantity error, but unfortunately, PAL will quietly assemble the wrong value. Instead of $def use $dee+1, or use $dee0+10 instead of $def0 when using either assembler.

SSS or Save SYMASS Symbols

Save SYMASS Symbols is a useful utility for machine language programming. It works with SYMASS, the symbolic assembler published last issue in The Transactor, to produce a disk file of a program's symbol table. You can load the file as a normal program, and list it on your screen or printer to help with debugging.

First, obtain a copy of SYMASS, type in the SSS source code, and assemble it. At this point, if you type:

```
sys820 " st.sss
```

. . .a PRG file called "st.sss" will be created on disk containing a list of every symbol and its value. This file starts with:

```
1 memsiz = $37
2 symptr = $52
3 symend = $57
```

When developing a large ML program, a symbol table will help you find unique names for routines and variables. SYMASS does NOT check for redefinition, allowing you to have multiple entries in the table under the same name. Use SSS to make sure you don't have that problem.

Since the symbol table is in program format, it can be modified using delete and renumber commands, such as the ones in TransBASIC's Prg Management module. Merge together a number of Kernal routines you use regularly, and you can merge their definitions into your new programs with TransBASIC's USE command.

Editor's Note

The SYMASS Assembler is available on Transactor Disk 12. For the most part, it is PAL compatible, except for some of PAL's more exotic features. Otherwise, SYMASS can be used to assemble virtually any machine language program published in The Transactor. The original source code is also on Disk 12 so you can see just how an assembler works. For a complete description of SYMASS, see Volume 7, Issue 01, page 69. SYMASS 3.1 is also included on The TransBASIC Disk.

SSS Source Code

```
FD 100 sys700
AM 110 ; <<< save symass symbols >>>
MG 120 ;                 1.3
DD 130 ;       robert huehn mar 1986
GG 140 *        =    820          ;820 to 1021
MA 150 ;
DD 160 memsiz   =    $37          ;top of memory
JL 170 symptr   =    $52          ;pointer to table
KL 180 symend   =    $57          ;bottom of table
PP 190 line     =    $39          ;line number
BO 200 clr      =    $a663        ;do clr
CG 210 eval     =    $ad9e        ;evaluate expression
AK 220 chkstr   =    $b6a3        ;check for string
DI 230 setnam   =    $ffbd        ;set file name
EP 240 setlfs   =    $ffba        ;set logical file
OH 250 open     =    $ffc0        ;open file
GJ 260 chkout   =    $ffc9        ;output channel
GH 270 chrout   =    $ffd2        ;print
OI 280 clrchn   =    $ffcc        ;clear channels
KO 290 close    =    $ffc3        ;close
OM 300 ready    =    $a474        ;ready
MK 310 ;
BB 320          lda  memsiz:sta symptr  ;protect
FC 330          lda  memsiz+1:sta symptr+1  ;symbol
HG 340          lda  symend:ldx symend+1  ;table from
II 350          jsr  lower
KC 360          jsr  eval       ;basic's evaluate
KN 370          jsr  chkstr     ;get length and pointer
FF 380          jsr  setnam     ;for kernal routine
NO 390          lda  #8         ;file 8, dev 8
LF 400          tax
NP 410          ldy  #1         ;sec addr 1
OG 420          jsr  setlfs
CC 430          jsr  open
PA 440          ldx  #8
EI 450          jsr  chkout
DC 460          lda  #1:jsr dout   ;fake start address
DO 470          lda  #0:sta line:sta line+1
IN 480          lda  symptr:ldx symptr+1
HH 490          jsr  lower      ;move memsiz back
GK 500 sss1     lda  symptr     ;check for end of
AH 510          sec             ;symbol table
FJ 520          sbc  #10
MN 530          sta  symptr
PG 540          bcs  sss2
PG 550          dec  symptr+1
GO 560 sss2     cmp  symend
II 570          lda  symptr+1
EL 580          sbc  symend+1
MN 590          bcc  done
HB 600          lda  #1:jsr dout
FF 610          inc  line       ;save fake link and new
IG 620          bne  sss3       ;line number
GO 630          inc  line+1
DP 640 sss3     lda  line:jsr chrout
PM 650          lda  line+1:jsr chrout
PN 660          ldy  #0
CK 670 sss4     lda  (symptr),y
FA 680          beq  sss5
KC 690          jsr  chrout     ;save symbol name
DE 700          iny  :cpy #8
KA 710          bne  sss4
AI 720 sss5     lda  #"  " :jsr chrout
AO 730          lda  #$b2:jsr chrout  ;save ' =$'
CP 740          lda  #" $ " :jsr chrout  ;and hex value
GP 750          ldy  #9:lda (symptr),y:beq sss6:jsr hex
EA 760 sss6     dey:lda (symptr),y:jsr hex
CO 770          lda  #0:jsr chrout
NH 780          jmp  sss1
BC 790 hex      =    *          ;print hex number
MJ 800          pha
MD 810          lsr:lsr:lsr:lsr
OG 820          jsr  he1
DB 830          pla:and #$0f
FF 840 he1      cmp  #10
DB 850          bcc  he2
FE 860          adc  #6
FM 870 he2      adc  #$30
NC 880          jmp  chrout
KA 890 lower    =    *
MG 900          sta  memsiz:stx memsiz+1
FO 910          jmp  clr
NB 920 dout     =    *
EG 930          jsr  chrout:jmp chrout
LF 940 done     lda  #0:jsr dout
OM 950          lda  #8          ;finish
FD 960          jsr  close
BN 970          jsr  clrchn
IM 980          jmp  ready
```

Transcribe 64

Richard Evers, Editor

A Relative File Copy Utility For The Commodore 64

Since the day that the Vic 20 made it's debut, programmers have purposely avoided relative files. The reasons given, if any, are usually the speed and reliability of the 1540/1541 serial drives and the regression to Basic 2.0 syntax. What was once a trusted friend among the Basic 4.0 Pet/CBM users had degenerated to a level of outcast among all. Even today, after scores of books and magazine articles have been written about the subject, fears persist.

In the face of these fears and misconceptions, some brave programmers have battled the odds and won. Many data base, accounting and communications systems have been written, some well – others not so well, making use of relative files as their main data storage medium. Although relative files have a few inherent limitations such as a maximum record length of 254 characters, a maximum number of records in a file of 65535 and a cap on the maximum amount of space available for use by a relative file on disk, good planning and a few programming tricks have been known to minimize the problems. And so, on to the purpose of this article.

The Purpose

Transcribe 64 is 1361 bytes of code that will do what no program has done before (the shadow of Star Trek looms overhead). It will single drive copy relative files using the Commodore 64 with any Commodore or compatible drive. After all these years, you can now duplicate all of your relative data files without being forced into bit copying the entire diskette each time. Although a little late, it has arrived.

As stated above, Transcribe 64 will copy relative files using any Commodore or compatible drive supporting them. If the drive requires that you use an IEEE interface then the interface might be the only stumbling block in your way. I tested the program using a GLINK IEEE interface to communicate with my 9090, 8250 and 2031 drives. Access to my 1571 (in 1541 mode) was made in the normal fashion. The GLINK is unique in that it uses no RAM whatsoever. It operates by swapping ROM depending on the setting of the serial/parallel switch. Transcribe 64, written entirely in assembler, uses all RAM from $1000–$FFFF except the $D000–$DFFF area (52k of storage). It also uses 6 bytes of zero page ($57–$5C) plus location $01 for swapping out RAM and ROM. Further to that, the input buffer ($0200–) is used to retrieve and modify the name of the file to be copied. And finally, the program itself occupies memory from the start of Basic up by 1361 bytes ($0801–$0D51). Other than that, memory is untouched. If your interface does not rely on RAM in any of the areas mentioned, you're okay. If not, you can either re–write the source, which shouldn't be too difficult, or change interfaces. Your choice if you are forced into making it.

The program has been written to perform as many 52k passes as required to copy the relative file chosen. Remember, though, that not all diskettes are created equal with regards to relative file storage space. The 1540, 1541, 2031 and 4040 can store a maximum of 658 blocks of relative data in one file. The 1571, in double sided mode from 64 mode, and the DOS 2.5 version of the 8050 can handle 720 blocks in total per file. The DOS 2.7 version of the 8050, after setting it up to handle expanded relative files, allows a maximum of about 508k of data per file. The DOS 2.7 8250 allows a maximum of about 1.04 megabytes of data per file. The Commodore 9060 and 9090 hard drives allow a maximum of 4.9 meg and 7.35 meg of data/relative file consecutively. As a fail safe rule when copying relative files remember, when in doubt, think. Imagine trying to fit 7.35 megabytes of data on a 167k, 1541 diskette. It could be tight. Error detection has been built into the program so that if a problem develops then all files will be closed up, the offending error number will be displayed and control will be passed back to Basic. Not too awful.

How It Works

To start, not all of the available RAM is used for the storage of relative data. One byte/record is consumed as a count of the number of characters read from each record. This method is employed due to the funny nature of relative files. If, for example, you have a relative record length of 40 characters but, after positioning to the first byte in the record, you only write 10 characters, you will find that you can only retrieve 10 characters from that record later. Regardless of what was in the record before the last write, DOS will prevent you from retrieving more than 10 characters. After reading in the tenth character, the variable ST will be set to a value of 64, end–of–file. If you persist in reading data beyond this point, the DOS will foil your foolish attempts and position to the next record automatically. One byte/record consumed in RAM is my method of compensating for the strange behavior of DOS.

The program has been written to simply Load and Run. No strange SYS addresses or special theatrics to perform before using. Once it is up and running, you will be prompted to place the source diskette in drive zero and enter the filename in which you wish to copy. Dual drive users beware: drive zero is the only one allowed for the occasion. When you enter the filename, you can enter it with or without a drive number. The length of the actual name entered is checked to make sure you don't exceed 16 characters.

Once you have supplied the filename, the program will Open and

read through the disks directory (via the equivalent of open 8,8,8, "$0"), looking specifically for relative file program types ($84). If found, a byte for byte check is made against the filename. If a match occurs and the filename length is less than 16 then the next byte in the directory filename is checked to see if it is a Shifted Space ie. CHR$(160). A complete filename in the disks directory is always followed by enough Shifted Spaces to pad in the complete length of 16. If the name has a length less than 16 then anything other than a CHR$(160) will signify that the match is incorrect.

Once the filename has been verified, the next three bytes following the filename are read in: the first side sector track + sector plus the relative record size. The record size is retained, the file is closed up and the program continues along its predetermined path.

With the record length known, one last variable remains to be found: the number of records in the file. To solve this problem, we need to open up the relative file for a quick pass through. First, a suffix of ",l," plus the record length is added to the filename with the length of the filename updated accordingly, then the file is opened up. Once open, a loop begins which positions to records #1 upwards until an error #50, no-record-present, is discovered via the command channel. With this error received, the record position is backed off 1 notch and the actual copy session is ready to begin.

The file is not closed after determining the highest record number. To save time, the copy session is entered directly. First, record #1 is positioned to and location $FFFF is marked as the first record character count address. Data is stored in RAM from $FFFF downwards. Following this, the data retrieved from the record is stored in RAM sequentially until ST=64. When this occurs, the record position is incremented, the new record is positioned to, error #50 is checked for to see if the end has come and, if not, the retrieval session begins again. Before actually bringing in the data each time, though, a quick bit of math is performed to see if: current RAM location - the record length - 1 (count) < lowest position in RAM storage. If not then the session continues.

Special Indicators

Two special indicators are used in lieu of the character count. They are available for use because the number of characters/record can never be equal to them. They are:

Byte Value $00: Indicates that the end of file has been reached and no more data is left to copy. When the write section of the copy routine discovers this value, it closes up all files, displays a 'copy complete!' message for the viewing audience and returns to Basic.

Byte Value $FF: When the read session runs out of available storage space, it flags a count of 255 to indicate that more data remains to be copied from the source file. When found by the write section, the destination file is closed up, a prompt is displayed asking that the diskettes be swapped and a wait until <carriage return> loop begins. Once <return> is pressed, the source file is re-opened and the read session starts up where it left off. Transcribe will make as many passes as necessary to copy the entire relative file.

The Write Session

The write session is really a read session in reverse. All data is pulled sequentially from RAM address $FFFF downwards, with the count of characters for each record used as an index for the write. There are no special tricks performed in this session. It simply gets the data and writes it to the correct record number. One note must be made about the write session, though. On the first pass, the destination file must be created before it can be accessed. Relative files are funny in this respect. Unlike sequential, user or program files that expand as you fill them, relative files need predetermined borders in which to operate. To create a relative file, you first open it for the first time with the correct record length then position to the maximum record number that you will need. From there you print CHR$(255); to that record. DOS then builds the file automatically. Once complete and the error channel reports that the creation went okay, the actual write session begins. Although the file could have been expanded, record by record, as the data presented itself without creating the entire file first, I felt it best to do it this way to make sure that the complete file would fit on disk. In providing this feature, the source and object was increased in size considerably. The final result seems to be worth the extra effort involved.

At this point I would like to mention two very special people who helped me with making this program come alive. While I was in the last throes of writing Transcribe, a rotten roadblock materialized with the 1541. Although the program worked perfect with my IEEE drives, the 1541 was having a tantrum. The program would repeatedly bomb out after creating the destination file for the first time. An error message number 4 (?) would be displayed each time after creation. At this stage, the 1541 would be in an error condition, flashing it's lights in a rather disturbing manner. By reading the error channel, a 'record-not-present' error was found. An odd state of affairs.

After repeated attempts to cure the problem, I did the obvious; I called Jim Butterfield. Jim, in his usual patient way, told me that I must have done something stupid. An error number 4 cannot happen in Basic, therefore, my coding was at fault. He proceeded to run through a few possible ways in which the code could be messing things up, of which only one rang a faint bell. After a few more minutes of idle chatter on my part, we hung up. The faint bell started to get louder. He had asked if I was reading ALL the data from the error channel until I reached a CHR$(13). I was not but, in my typical myopic manner, I could not see how my IEEE drives could live with an incompletely read error channel when the 1541 could not. To quickly finish up, Jim was right. After modifying the code to read the entire message, the 1541 was happy. Jim B. to the rescue once again.

The program was finally working just fine, so it was time to phone Chris Zamara to boast a little. At the time he was busily putting together his 'Animals' program which relies on relative files for record keeping. During our conversation I mentioned that one thing was still bothering me after completing the code; would the 1541 reliably write the data each time. I had tested it quite a bit but still did not completely trust it. After so many years of bad press, old fears die hard. Chris came up with an article in Compute! magazine from back in July 1985 that stated that you should always position to a record, write your data, then position once again to the same record to prevent 'spill-over'. It sounded odd but it seemed to work for Chris. I had been positioning twice to each record before a read or write, just to be on the safe side. It was now

clear that I was in the dark. Another phone call was in order.

On the conclusion of our conversation, I called up Dr. Gerald Neufeld, the co-author of 'Inside Commodore DOS' (Datamost ISBN 0-88190-366-3) and author of the '1541 User's Guide' (Datamost ISBN 0-88190-396-5). If a reliable answer was needed, why not go to the source?

Gerald was terrific. We talked, his grandaughter played in the background, and I learned that for sequential access to relative files, no special tricks need be performed. A single position and a write would work just fine. If writing in a random sequence to records was required, though, the record should be positioned to twice, a time delay of about .5 second should be imposed then the write could be performed safely. After a few more minutes of conversation, we said goodbye and I began removing all the duplications of positioning within my code. Needless to say, it still worked just fine. Transcribe 64 was a completed commodity.

Before finishing up, I should mention a few points that I gleaned from the '1541 User's Guide' and from Jim Butterfield. First, three record sizes are not allowed when creating relative files. They are 42, 58 and 63. Apparently, Gerald discovered this after many days of exhaustive testing with the 1541. He figured that DOS gets confused with the characters '*' CHR$(42), ':' CHR$(59) and '?' CHR$(63). Further to this he states that the position within the record statement, ie.:

```
print#15," p " chr$(SA OR 96)chr$(Rec# Low)chr$(Rec# Hi)chr$(Pos)
```

. . .is definitely not an option. Problems will develop when left off. When it was used, though, he found it to be reliable. From Jim B. I learned not to create a relative file that takes up less than one sector (254) of data. Apparently the side sector information will not be written, therebye creating a mess. It's so nice to get good information.

In conclusion, a little not so great news. We are not printing the source for Transcribe 64 this issue mainly because it measures in around 16k. Although the program is quite useful, we have so much more to print that my source gets last priority. It will be included on The Transactor Disk #13 along with the Load and Run module Transcribe 64. For those of you so inclined, my code has been written using only one non-kernel routine, Input. The source is well commented throughout and has been written in PAL format. For all intentions, I now consider the program and source to be in public domain and I wish you all the best of luck with it. If you make modifications to the source code and feel it worthy of general use, please drop us a copy so we can include it on our Transactor disks. If you make a revision, though, make sure to update the version number and, in the least, put your name and date in the source listing. This will help when determining the latest update.

Transcribe 64: BASIC Loader

```
DL 1000 rem save " 0:transcribe64.bas " ,8
JI 1010 rem ** transcribe 64: rte/86
PB 1020 rem **  a relative file copy prg
NP 1030 rem **  for the commodore 64
EI 1040 :
CI 1050 rem ** this program will create
PN 1060 rem ** a load and run module on
FJ 1070 rem ** disk called 'transcribe 64'
MK 1080 :
DE 1090 open 15,8,15: open 8,8,1, " 0:transcribe 64 "
BN 1100 input#15,e,e$,b,c: if e then close 15:
        print e;e$;b;c: stop
DD 1110 for j = 2049 to 3411: read x: print#8,chr$(x);:
        ch = ch + x: next: close8
BH 1120 if ch<>136167 then print " checksum
        error ! " : stop
LC 1130 print " ** module created ** " : end
IO 1140 :
KP 1150 data   1,   8,  11,   8,  64,   0, 158,  50
KC 1160 data  48,  54,  49,   0,   0,   0, 162,   0
EG 1170 data  32,  87,  12,  32,  96, 165, 162,   0
BL 1180 data 189,   0,   2, 240,   3, 232, 208, 248
ND 1190 data 169,  17, 141,  64,  13, 173,   1,   2
BN 1200 data 201,  58, 208,   5, 169,  19, 141,  64
NN 1210 data  13, 236,  64,  13, 176, 216, 142,  56
PA 1220 data  13, 169,  15, 162,   8, 160,  15,  32
OP 1230 data 186, 255, 169,   0,  32, 189, 255,  32
JB 1240 data 192, 255,  32, 185,  11, 192,   0, 240
OP 1250 data   3,  76, 207,  11, 169,   8, 162,   8
PN 1260 data 160,   8,  32, 186, 255, 169,   2, 162
JC 1270 data  54, 160,  13,  32, 189, 255,  32, 192
MC 1280 data 255,  32, 119,  11, 192,   0, 240,   3
KE 1290 data  76, 207,  11, 162,   8,  32, 198, 255
HA 1300 data 160,   8, 162,  32, 142,  64,  13, 192
JJ 1310 data   1, 208,   5, 162,  30, 142,  64,  13
EI 1320 data 206,  64,  13,  32, 207, 255, 201, 132
IJ 1330 data 208, 106, 206,  64,  13,  32, 207, 255
OE 1340 data 206,  64,  13,  32, 207, 255, 152,  72
KD 1350 data 160,   0, 140,  65,  13, 173,   1,   2
KJ 1360 data 201,  58, 208,   2, 160,   2, 206,  64
KF 1370 data  13,  32, 207, 255, 217,   0,   2, 208
MG 1380 data  65, 238,  65,  13, 200, 204,  56,  13
GP 1390 data 208, 236, 172,  65,  13, 192,  16, 240
IJ 1400 data  25, 206,  64,  13,  32, 207, 255, 201
AM 1410 data 160, 208,  39, 240,   6,  32, 207, 255
BJ 1420 data 238,  65,  13, 172,  65,  13, 192,  15
NM 1430 data 208, 243,  32, 207, 255,  32, 207, 255
HM 1440 data  32, 207, 255, 141,  60,  13,  32, 204
ON 1450 data 255, 169,   8,  32, 195, 255, 104,  76
BM 1460 data  30,   9, 104, 168,  32, 207, 255, 206
BD 1470 data  64,  13, 208, 248,  32, 183, 255, 240
EM 1480 data  13,  32, 204, 255, 169,   8,  32, 195
LB 1490 data 255, 162,  84,  76,  87,  12, 136, 208
EB 1500 data   3,  76, 119,   8,  76, 121,   8, 174
BD 1510 data  56,  13, 160,   0, 185,  57,  13, 157
EP 1520 data   0,   2, 232, 200, 192,   3, 208, 244
OP 1530 data 173,  60,  13, 157,   0,   2, 232, 142
KA 1540 data  56,  13,  32, 230,  11, 192,   0, 240
OH 1550 data   3,  76, 207,  11, 169,   1, 141,  69
KE 1560 data  13, 169,   0, 141,  70,  13,  32,   0
BF 1570 data  12, 192,   0, 240,   7, 192,  50, 240
CD 1580 data  13,  76, 207,  11, 238,  69,  13, 208
PK 1590 data 237, 238,  70,  13, 208, 232, 206,  69
DD 1600 data  13, 173,  69,  13, 201, 255, 208,   3
HC 1610 data 206,  70,  13, 141,  79,  13, 173,  70
PG 1620 data  13, 141,  80,  13, 169,   1, 141,  81
KI 1630 data  13, 165,   1, 141,  77,  13,  41, 252
HL 1640 data 141,  76,  13, 169,   1, 141,  74,  13
```

```
EL 1650 data 141,  72,  13, 169,   0, 141,  75,  13
HL 1660 data 141,  73,  13, 169,   0, 141,  63,  13
MI 1670 data 173,  72,  13, 141,  69,  13, 173,  73
ML 1680 data  13, 141,  70,  13, 169,   0, 133,  87
JB 1690 data 169, 255, 133,  88, 160, 255, 140,  65
JK 1700 data  13, 165,  88, 133,  92, 172,  65,  13
CM 1710 data 132,  91,  32,  19,  11, 224,   2, 208
MP 1720 data   9, 169, 255, 160,   0, 145,  91,  76
HJ 1730 data  48,  10, 165,  88, 201,  16, 208,  11
PN 1740 data 173,  65,  13,  56, 237,  60,  13, 233
EP 1750 data   1, 144, 230, 169,   0, 141,  78,  13
AE 1760 data  32,   0,  12, 192,   0, 240,  15, 192
OF 1770 data  50, 208,   8, 169,   0, 168, 145,  91
EK 1780 data  76,  48,  10,  76, 207,  11, 162,   1
FL 1790 data  32, 198, 255,  32, 207, 255,  72, 238
GP 1800 data  78,  13,  32, 183, 255, 141,  63,  13
KC 1810 data  32, 204, 255, 104,  32,  12,  11, 224
JB 1820 data   0, 240, 227, 224,   1, 240, 170, 173
IJ 1830 data  78,  13, 160,   0, 145,  91, 238,  69
GJ 1840 data  13, 208, 142, 238,  70,  13,  76, 184
PM 1850 data   9, 169,   1,  32, 195, 255, 173,  69
IC 1860 data  13, 141,  72,  13, 173,  70,  13, 141
GG 1870 data  73,  13, 162, 103,  32,  87,  12,  32
BC 1880 data 107,  12, 173,  81,  13, 240,  10,  32
LN 1890 data  25,  12, 192,   0, 240,   3,  76, 207
OG 1900 data  11, 173,  74,  13, 141,  69,  13, 173
HJ 1910 data  75,  13, 141,  70,  13, 169, 255, 133
EO 1920 data  90, 169,   0, 133,  89, 160, 255, 140
KJ 1930 data  65,  13,  32, 185,  11,  32, 230,  11
GM 1940 data 192,   0, 240,   3,  76, 207,  11, 165
DK 1950 data  90, 133,  92, 169,   0, 133,  91, 120
PP 1960 data 174,  76,  13, 134,   1, 172,  65,  13
NB 1970 data 177,  91, 174,  77,  13, 134,   1,  88
EJ 1980 data 201,   0, 240,  71, 201, 255, 208,  47
CM 1990 data 173,  72,  13, 141,  69,  13, 141,  74
BM 2000 data  13, 173,  73,  13, 141,  70,  13, 141
KB 2010 data  75,  13, 169,   1,  32, 195, 255, 162
OO 2020 data 103,  32,  87,  12,  32, 107,  12,  32
DK 2030 data 185,  11,  32, 230,  11, 192,   0, 240
LJ 2040 data   3,  76, 207,  11,  76, 154,   9, 141
BA 2050 data  78,  13,  32,  88,  11,  32,   0,  12
PA 2060 data 192,   0, 240,  22, 192,  50, 240,   3
IF 2070 data  76, 207,  11, 169,   1,  32, 195, 255
NK 2080 data 169,  15,  32, 195, 255, 162, 164,  76
EG 2090 data  87,  12, 162,   1,  32, 201, 255,  32
JB 2100 data  55,  11, 206,  78,  13, 208, 248,  32
LH 2110 data 204, 255, 238,  69,  13, 208,   3, 238
NE 2120 data  70,  13,  76, 126,  10, 162,   0, 172
CG 2130 data  65,  13, 145,  87, 206,  65,  13, 136
EF 2140 data 192, 255, 208,  18, 198,  88, 165,  88
DN 2150 data 201, 223, 208,   6, 169, 207, 133,  88
MI 2160 data 208,   4, 201,  15, 240,   8, 173,  63
GH 2170 data  13, 201,  64, 208,   2, 232, 232,  96
FJ 2180 data 172,  65,  13, 120, 173,  76,  13, 133
BO 2190 data   1, 177,  89,  72, 173,  77,  13, 133
LK 2200 data   1,  88,  32,  88,  11, 104, 224,   0
EI 2210 data 240,   3,  76, 207,  11,  32, 210, 255
JO 2220 data  96, 162,   0, 206,  65,  13, 136, 192
JE 2230 data 255, 208,  20, 198,  90, 165,  90, 201
LC 2240 data 223, 208,   6, 169, 207, 133,  90, 208
OO 2250 data   6, 201,  15, 208,   2, 162,   1,  96
AG 2260 data 162,  15,  32, 198, 255,  32, 207, 255
FO 2270 data 141,  61,  13,  32, 207, 255, 141,  62
BE 2280 data  13,  32, 207, 255, 201,  13, 208, 249
ID 2290 data  32, 204, 255, 173,  61,  13,  56, 233
BF 2300 data  48, 141,  66,  13,  24, 162,   9, 109
EP 2310 data  66,  13, 202, 208, 250, 141,  66,  13
DC 2320 data 173,  62,  13,  56, 233,  48,  24, 109
IA 2330 data  66,  13, 168, 192,  20, 176,   2, 160
CC 2340 data   0,  96, 162,  15,  32, 201, 255, 169
MD 2350 data  73,  32, 210, 255, 169,  48,  32, 210
AI 2360 data 255,  32, 204, 255,  32, 119,  11,  96
BM 2370 data 169,  15,  32, 195, 255, 162, 182,  32
EE 2380 data  87,  12, 173,  61,  13,  32, 210, 255
AJ 2390 data 173,  62,  13,  32, 210, 255,  96, 169
AH 2400 data   1, 162,   8, 160,  12,  32, 186, 255
DP 2410 data 173,  56,  13, 162,   0, 160,   2,  32
DD 2420 data 189, 255,  32, 192, 255,  32, 119,  11
CG 2430 data  96, 162,  15,  32, 201, 255, 162,   0
PA 2440 data 189,  67,  13,  32, 210, 255, 232, 224
KI 2450 data   5, 208, 245,  32, 204, 255,  32, 119
IM 2460 data  11,  96, 162, 138,  32,  87,  12,  32
AO 2470 data 185,  11, 192,   0, 240,   1,  96, 173
GP 2480 data  79,  13, 141,  69,  13, 173,  80,  13
MB 2490 data 141,  70,  13,  32, 230,  11,  32,   0
GP 2500 data  12, 162,   1,  32, 201, 255, 169, 255
OD 2510 data  32, 210, 255,  32, 204, 255,  32, 119
MP 2520 data  11, 152,  72, 169,   1,  32, 195, 255
KM 2530 data 104, 168, 169,   0, 141,  81,  13,  96
DE 2540 data 169,  13,  32, 210, 255,  32, 210, 255
OB 2550 data 189, 115,  12, 240,   6,  32, 210, 255
EC 2560 data 232, 208, 245,  96,  32, 228, 255, 201
MK 2570 data  13, 208, 249,  96,  82,  69,  76,  65
EH 2580 data  84,  73,  86,  69,  32,  70,  73,  76
AK 2590 data  69,  32,  67,  79,  80,  73,  69,  82
NI 2600 data  32,  86,  49,  46,  48,  48,  32,  45
AI 2610 data  32,  82,  84,  69,  47,  56,  54,  13
HK 2620 data  13,  18,  32,  80,  76,  65,  67,  69
DL 2630 data  32,  83,  79,  85,  82,  67,  69,  32
DL 2640 data  68,  73,  83,  75,  32,  73,  78,  32
PN 2650 data  68,  82,  73,  86,  69,  32,  90,  69
FK 2660 data  82,  79,  32,  13,  13,  70,  73,  76
JL 2670 data  69,  78,  65,  77,  69,  32,  63,   0
KO 2680 data  18,  32,  70,  73,  76,  69,  32,  78
HC 2690 data  79,  84,  32,  70,  79,  85,  78,  68
JK 2700 data  33,  32,   0,  18,  32,  83,  87,  65
KA 2710 data  80,  32,  68,  73,  83,  75,  69,  84
OA 2720 data  84,  69,  83,  32,  58,  32,  80,  82
HB 2730 data  69,  83,  83,  32,  60,  82,  69,  84
NO 2740 data  85,  82,  78,  62,  32,   0,  67,  82
JE 2750 data  69,  65,  84,  73,  78,  71,  32,  68
DG 2760 data  69,  83,  84,  73,  78,  65,  84,  73
LB 2770 data  79,  78,  32,  70,  73,  76,  69,   0
LE 2780 data  18,  32,  67,  79,  80,  89,  32,  67
DI 2790 data  79,  77,  80,  76,  69,  84,  69,  33
GC 2800 data  32,   0,  68,  73,  83,  75,  32,  69
EC 2810 data  82,  82,  79,  82,  32,  35,   0,  36
HE 2820 data  48,   0,  44,  76,  44,   0,   0,   0
MN 2830 data   0,   0,   0,   0,  80, 108,   1,   0
LO 2840 data   1,   1,   0,   1,   0,   0,   0,   0
AI 2850 data   0,   0,   0
```

Animals:
An Exercise In Artificial Intelligence

Chris Zamara, Technical Editor

This program is presented as a game, but it uses principles of artificial intelligence that could make it useful in all sorts of interesting ways.

Programs like this have appeared in publications in the past, including the books "What to do after you press RETURN" from People's Computing, and David Ahl's "101 BASIC Games". As far as I know, it may have been around on mainframes long before that, so this is nothing new: just an implementation which will run on the PET/64/VIC/+4/B128 etc., and some ideas about applications. The first time I heard about the "animals" program concept was in Jim Butterfield's article called "Artificial (Fake?) Intelligence" in a recent TPUG magazine. Jim's mention of animals inspired me to write the program presented here.

Put simply, Animals is a Database which increases its knowledge as it is used, and locates records in the Database by asking specific questions. It is called animals because in its simple form as presented here, it can be used as a "guess the animal" game, where the computer tries to guess an animal that you're thinking of by asking questions. To begin with, the program knows only two animals and one question, and builds its knowledge of animals as it is used, learning a new animal each time it makes a wrong guess. To illustrate how it does this, take a look at this sample session. The only thing the program knows at first is the question "Does it live in the water?" and the animals "Fish" and "Horse".

```
Does it live in the water? no
  *** It might be a Horse
Is that correct (y/n)? yes

Alright! guess I'm pretty smart! Like to play again?
```

In the above case, the player was coincidentally thinking of one of the animals the computer knew, but let's try again, thinking of a bird.

```
Does it live in the water? no
  *** It might be a Horse
Is that correct (y/n)? no

Ok, what were you actually thinking of?
  Bird

  What yes/no question could I ask to distinguish a Horse from
  a Bird?
? Does it fly
  And regarding a horse, Does it fly? no
  Thank you for teaching me a new animal!

Play again?
```

The program has now learned a new animal, and will ask "Does it fly" whenever it get a "no" to "Does it live in the water?", and guess "bird" or "horse" depending on the answer. As the program is used more and more, it learns new animals, and can "intelligently" guess many animals correctly if the people who teach it give good questions. The interesting thing about this program is that the programmer who wrote it may know little more than the animals Horse and Fish, but after his program has been used by others for awhile, it may know hundreds of animals that he's never heard of. (The animals and questions are stored on disk in a relative file, independent of the program itself.) Left in a classroom, or better yet, with a few zoologists, the program would be quite well-educated, as it would assimilate the sum knowledge of all who used it.

After a bit more use, a sample session with the same program might go something like this:

```
Does it live in the water? no
Does it fly? no
Can you ride it? no
Does it have soft fur? no
Is it covered with hair? no
Does it have legs? no
Does it tough-skinned? yes
Does it eat ants? yes

It might be an aardvark. Is that correct? yes
```

The program gets smarter as it is used, but only when given questions which properly distinguish the two animals. Also, the questions given are most useful when they split the "yes" and "no" answers into nearly equal groups. For example, If you were to give a question to distinguish a dog from a chicken, "Does it bark?" would be not much better than "Is it a dog", since so many more possibilities exist on the "no" side than the "yes". A better question in this case would be "Does it have feathers?", or even "Does it lay eggs?", since such questions give a more even split and will lead to the final answer quicker. The system is basically a binary tree, where each branch is a question that leads to two more questions or answers. Choosing good questions will result in an evenly distributed tree.

Another thing to keep in mind when teaching the program a new animal is that once you reach the end of a branch of the tree, there is no way to add on to that branch. Because of that, you can't teach the program an animal like "bird", and then later want to get more specific and teach it "sparrow" or "robin". Once it guesses bird, it'll be right or wrong, and if it's right, you've reached the end of the line. If you want to have a more specific database, you could teach it a sparrow and use "is it a bird" as the distinguishing question.

A good use of the program, which gets around the above problem, is to use it to guess specific people instead of animals. Questions like "is he/she female?", "Does he/she have brown hair?", etc. could be used to lead to the names of your friends. Each of your friends could in turn introduce new people to the program, and before long you'll have hundreds of people in your database, and you would be able to find a the name (and perhaps even phone number) of someone simply by answering the computer's questions about their physical appearance or likes and dislikes.

Notice how we've just found a completely different use for the same program, simply by giving it different information. In fact, you could have a "people" as well as an "animals" data file on disk, and use the identical program, since it asks you the filename of the data when you run it. You could build up lots of files containing different types of information, and then just RUN the program and tell it "animals", "people", "aircraft", "famous people", or whatever you care to find out about. From a description of an airplane or an actor, you could find out the identity, based on what the program knows so far.

If you do start using the program to build large databases, you'll find that it has limitations which make it difficult to use in a practical way. For example, if someone puts in a bad or incorrect question somewhere along the way, there's no easy way to repair the damage, making the database unreliable. Ambiguous questions could end up with an animal (or whatever things you're using) ending up in two different places in the tree. And a tree that started with specific instead of more general questions (like asking "Is it a marsupial" right off) would need to ask many questions before getting to the end of a branch.

By expanding on the program, all of the above drawbacks could be addressed. The program could check the tree for animals in two places, and ask questions to find the correct place. The branches in the tree could be re-arranged, putting the most common questions at the beginning. It could check whether a new animal being entered is correct based on all previous questions in the path, and ask questions to make the necessary corrections if a discrepancy is found. Branches of the tree which are never accessed could be labelled as possibly "bad", and the program could determine which questions to ask to fix it up. And the problem of expanding beyond the end of a branch could be solved by allowing multiple-choice answers instead of just yes or no, for example, "it could be a: 1)sparrow, 2) robin, 3)duck".

An animals program incorporating all of those features could be used as a kind of "expert system", learning from those who use it and providing answers to questions based on observed facts. As an example, let's use Animals as a car problem diagnosing tool. At first, all it knows is the question "Does the engine turn over?" and the answers "Check the ignition cap", and "Check the battery". You find your car doesn't turn over and the program tells you to check the battery. You find that the problem isn't the battery, go to a mechanic, and he finds a cooked starter solenoid. "I thought it was the battery", you say. "No", says the mechanic, "Look how bright the headlights are, and they don't even dim when you try to start – obviously, it's the starter or the solenoid." After visiting your friendly local loans officer so that you can pay the mechanic, you head home to teach "Animals" something new. When it says "Try the battery. Is that correct?", you say "no", and tell it about the solenoid. When it asks how to distinguish a weak battery from a cooked solenoid, you say, "Do the headlights dim when attempting to start?", and say "no" for the bad solenoid. Now, much to the dismay of your mechanic and bank manager, you are on your way to building a complete car-maintenance database.

If you were to use the program for the life of your car, it might learn a few things, but nothing that you didn't learn yourself. But if the program was properly used at, say, a car dealership's service department, the dumb little animals program would have the experience of a dozen mechanics at its disposal. Using the 1541 drive, you could fit over 1,700 questions and diagnoses onto a disk – the number is only limited by disk space. You could actually do this with the program listed here, but you may be frustrated if you mess up the database somehow. There are, however, expert systems for microcomputers that use a similar concept, and are designed for use in everyday situations. This program is just for fun, and to show how easy it is to achieve a "artificial intelligence" on a simple level.

Before you RUN the "Animals" program in the following listing, you'll have to create the relative file containing the database. Part of the program is set up to do this for you: RUN 51000 to create the file. You will be asked the filename (the default is "animals.dat"), and the maximum number of records in the file. 2000 is a good big number, creating a file which takes 358 blocks on disk. It will take a short while for this file to be created. Once created, the first question and the animals fish and horse will be written to the file to start you off. You can change the start question and answers in lines 51110 to 51130. If at any time, you wish to re-start the database without re-creating the file, just RUN 51100. To copy a relative file used by Animals, you can use the program "Transcribe", from page 42 in this issue.

Once the file is created, just RUN the program and enter the database filename or press RETURN to use "animals.dat". From there on, you can use animals as described above. To answer the yes/no questions, just press 'y' or 'n' without RETURN. The program has been tested on an 8032, C-64, and B500, and on both the 8050 and 1541 drives. It should work on the VIC, +4 and C-128 as well.

When using the program, please note that no disk error checking is done. The program is all in BASIC, so you can easily add error-checking yourself. The program was kept minimal to make it easy to type in.

How The Program Works

For all its seeming brilliance, "Animals" is a very simple program, and was quite easy to write. Each question or animal is stored as a record in a relative file. The first four bytes of each record store two pointers, which are all zero in the case of an animal, and point to other records in the case of a question. A Question's first pointer points to the record read for a "yes" answer, and the second is for a "no". Adding a new question merely consists of inserting it in this chain, between a question and an animal, and making it point to the appropriate animals.

For example, when the program first starts up, there are three records filled (actually 4, since record #1 holds a pointer to the next

available record). Record #2 contains the question "Does it live in the water" and the "yes" pointer points to record #3, "Fish", and the "no" pointer points to "Horse" in record #4. If you are thinking of a bird, you answer "no" to the question and tell it "Bird". You then give it the question "Does it fly" and say for a birds the answer is "yes". The program then changes the "no" pointer of "Does it live in the water" to point to "Does it fly", the "yes" pointer of "Does it fly" points to the new animal, "Bird", and the "no" pointer points to the animal which previously ended the branch, "Horse". A new question has been inserted between a question and an animal, with the old and new animal forming the two branches of the new question. Record 5 now holds the new question, 6 the new animal, and the next–record pointer is set to 7. To see how the file looks, RUN 50000 to display the file along with the record numbers and pointers. You'll see the questions and where they point to, and how questions and animals are stored in alternate records.

When the program restarts after guessing correctly, it chains through each question by following the "yes" or "no" pointers depending on the answers to the questions. When an animal is reached and it is incorrect, the above process is followed again to insert a new question and animal. Simple, straightforward code that an intermediate programmer could produce with little effort.

If nothing else, this program may de–mystify the concepts of programs that learn by their errors and appear to be "smart". It is a lot of fun to play with, and may actually be useful as a simple expert system. Give it a try and mystify your friends with "fake" intelligence!

Thanks goes to Jim Butterfield for his insights and his article "Artificial (Fake?) Intelligence", which inspired this program.

Animals (Please note: some lines were altered slightly just before printing. Verifizer codes for these lines will show as "––".)

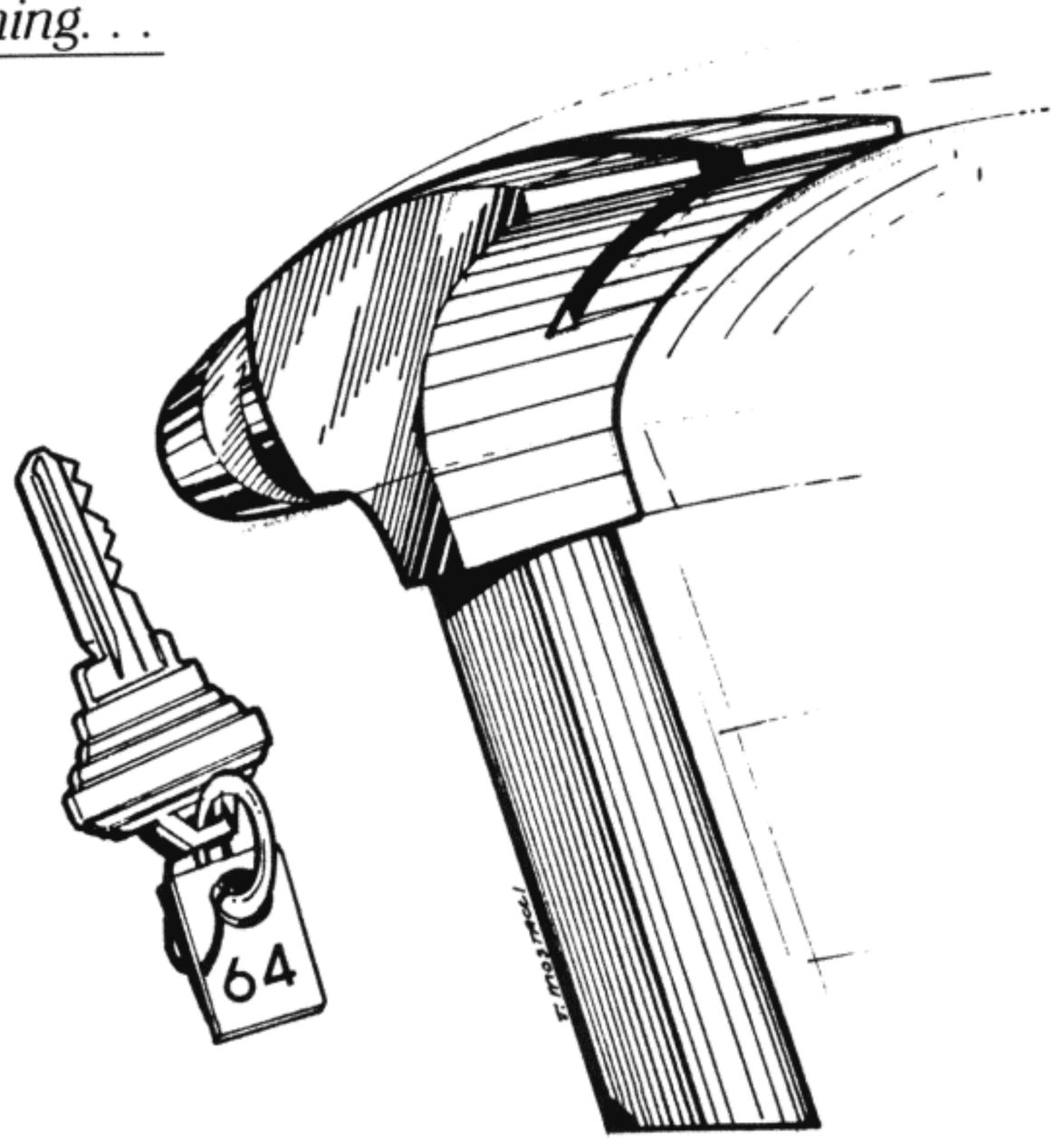
```
AB  100 rem  "animals" ai program
CE  110 rem  a simple expert system
HB  120 rem  run 51000 to create file
EF  130 rem  run 51100 to initialize file
LP  140 rem  run 50000 to print file
PI  150 rem   save"@0:animals 2.0",8
EB  160 :
NN  170 z$=chr$(0)
GD  180 sp$="        [56 spaces]        "
LP  190 rl=44: rem  rel record size–1
AO  200 input"Name of data file[4 spcs]animals.dat
        [13 lefts]";f$
GE  210 :
OB  220 rem* main program loop *
EK  230 open 15,8,15
KO  240 open 1,8,9,f$
MJ  250 print#15,"p";chr$(9)chr$(1)chr$(0)chr$(1)
ID  260 rem  first record holds next available record
DE  270 get#1,m1$,m2$: rem  in low, hi format
AN  280 m1$=left$(m1$+z$,1): m2$=left$(m2$+z$,1)
OI  290 max=asc(m1$) + 256*asc(m2$)
AK  300 :
IJ  310 print"■S■**[8 spcs]Think of an
        animal[10 spcs]**";
OI  320 print"**   Answer questions with 'y' or
        'n'   **■qq■"
NG  330 rp=2: rem  point to first question
IM  340 :
KI  350 r=rp: gosub 20000 'read in data
HA  360 if yes=0 and no=0 then 460 'end of chain
OB  370 rem  chain to next branch
EC  380 print m$;"?  ";
KP  390 gosub 10000: rem  get y/n response
JG  400 bp=rp: remember old record #
LE  410 if yn$="y" then a$="yes": rp=yes: ob=0
OJ  420 if yn$="n" then a$="no" : rp=no : ob=1
IB  430 print a$: rem  yes or no
HG  440 goto 350
GD  450 :
LF  460 rem end of chain – give guess
PA  470 print"■q■*** It might be a ";m$
HB  480 print"> Is that correct (y/n)  ";
JC  490 gosub 10000 'get answer yes or no
PP  500 if yn$="y" then 950 'found answer, wrap up
OC  510 rem got wrong answer, let's learn from it
EL  520 print"  no": print"■q■OK, what were you actually
        thinking of"
HA  530 input"  >";animal$
FG  540 print"■q■What yes/no question could I ask"
NH  550 print"to distinguish a"
BJ  560 print"     ";m$
HO  570 print"from a"
CH  580 print"     ";animal$
LN  590 input q$
EO  600 print"■q■And regarding a"
OB  610 print"     ";m$;",": print q$
JL  620 print">   (y/n)";
DE  630 gosub 10000 'get yes/no
CM  640 rem create new question pointing to current or
        new animal
JD  650 if yn$="y" then a$="yes": yn=rp:
        nn=max+1: rem new yes/no pointers
DI  660 if yn$="n" then a$="no" : yn=max+1:
        nn=rp: rem new yes/no pointers
IA  670 print a$: rem  yes or no
––  680 rr=max: gosub 40000 'record#(max)
DG  690 n=yn: gosub 30000: yn$=lh$: rem  convert n to
        lh$ (low+hi)
LC  700 n=nn: gosub 30000: nn$=lh$: rem  convert n to
        lh$ (low+hi)
JB  710 print#1,yn$;nn$;left$(q$+sp$,rl–4)
PP  720 gosub 40000 're–position to foil bug
HG  730 rem  point old question to new
LI  740 r=bp : gosub 20000 'read in old question
––  750 n=max: gosub 30000 'find low,hi of new
        record position
GL  760 if ob=0 then yes$=lh$: rem  point 'yes' ptr to
        new question
FN  770 if ob=1 then no$ =lh$: rem  point 'no' ptr to
        new question
––  780 rr=bp: gosub 40000 'record#1,(bp)
AH  790 print#1,yes$;no$;m$: rem re–write modified
        record
PE  800 gosub 40000 're–position to foil bug
AL  810 rem  now put new animal in next available record
```

```
--  820 rr = max + 1: gosub 40000 'record#1,(max + 1)
FL  830 print#1,z$;z$;z$;z$;left$(animal$ + sp$,rl-4)
HH  840 gosub 40000 're-position to foil bug
IE  850 rem  now update max. record pointer
KM  860 rr = 1: gosub 40000 '1st rec has ptr
DE  870 max = max + 2 : rem  2 records have been added
        to file
OK  880 n = max: gosub 30000 'convert to 2-byte pointer
KD  890 print#1,lh$;left$(sp$,rl-2): rem  pad with spaces
DL  900 gosub 40000 're-position to foil bug
CO  910 print: print " q Thank you for teaching me a
        new animal! "
BG  920 goto 970
GB  930 :
LI  940 rem got right answer, wrap up
FN  950 print "  yes " : print " q Alright! Guess I'm pretty
        smart. "
ED  960 :
GC  970 close 1: close 15
DM  980 input " qq play again[3 spcs]y[3 lefts] " ;yn$
LI  990 if yn$ = " y "  then 230
IO  1000 end
GG  1010 :
AH  1020 :
CF  10000 rem* subroutine to accept y or n
AH  10010 k = 0: for i = 0 to 1
NM  10020 get yn$
HF  10030 rem  flash fake cursor
PD  10040 print mid$(" rR " ,sgn(k and 8) + 1,1);
            " [1 spc, 1 crsr left] " ;
KO  10050 k = (k + 1) and 255
EO  10060 i = -(yn$ = " y " or yn$ = " n " ): next
DF  10070 rem  until 'y' or 'n' pressed
DL  10080 print " R   " ;: rem erase cursor
GI  10090 return
IO  10100 :
CP  10110 :
HC  20000 rem subroutine to read record# (r) in
            yes$,no$,m$
PD  20010 rr = r: gosub 40000: rem  record#1,(r)
BI  20020 get#1,y1$,y2$,n1$,n2$
NJ  20030 yes$ = left$(y1$ + z$,1) + left$(y2$ + z$,1)
CG  20040 no$ = left$(n1$ + z$,1) + left$(n2$ + z$,1)
PH  20050 yes = asc(y1$ + z$) + 256*asc(y2$ + z$)
BN  20060 no = asc(n1$ + z$) + 256*asc(n2$ + z$)
GE  20070 input#1,m$
CO  20080 rem  strip trailing spaces
HE  20090 lc = 0: for k = 1tolen(m$):if mid$(m$,k,1)<>
            "  " then lc = k
LL  20100 next: if lc then m$ = left$(m$,lc)
KK  20110 return
MA  20120 :
EF  30000 rem* subroutine to convert 16-bit 'n' to
            low,hi 'lh$' *
FE  30010 hh% = n/256: ll% = n-256*hh%:
            lh$ = chr$(ll%) + chr$(hh%)
AG  30020 return
CM  30030 :
MM  30040 :
JI  40000 rem* subroutine to simulate 'record#1,(rr)' using
            basic 2.0 *
PN  40010 rh% = rr/256: rl% = rr-256*rh%
NL  40020 print#15, " p " ;chr$(96 + 9)chr$(rl%)chr$(rh%)
            chr$(1)
KH  40030 return
MN  40040 :
GO  40050 :
JD  50000 rem**  dump relative file
IB  50010 open 15,8,15
MH  50020 input " Name of data file[4 spcs]animals.dat
            [13 lefts] " ;f$
IG  50030 open 1,8,9,f$
LK  50040 z$ = chr$(0)
MC  50050 print#15, " p " ;chr$(9)chr$(1)chr$(0)chr$(1):
            get#1,l$,h$
LF  50060 nr = asc(l$ + z$) + 256*asc(h$ + z$)-1
KL  50070 print "   1  : " nr " records in file q "
MF  50080 rl = 2: rh = 0: rem  record #, lo/hi
LP  50090 for i = 2 to nr
AO  50100 print#15, " p " ;chr$(9)chr$(rl)chr$(rh)chr$(1)
JE  50110 get#1,x1$,x2$,x3$,x4$: input#1,a$
PA  50120 p1 = asc(x1$ + z$) + 256*asc(x2$ + z$)
IC  50130 p2 = asc(x3$ + z$) + 256*asc(x4$ + z$)
IM  50140 print i; " : " ;p1,p2, " [ " a$ " ] "
IJ  50150 rl = rl + 1: if rl>255 then rl = 0: rh = rh + 1
NL  50160 next: close 15
KP  50170 end
IH  50180 :
CI  50190 :
EH  51000 rem**  create new animal file
KF  51010 input " Name of data file[4 spcs]animals.dat
            [13 lefts] " ;f$
IJ  51020 input " maximum number of records[6 spcs]
            2000[6 lefts] " ;m
EB  51030 open 15,8,15
DB  51040 open 1,8,9,f$ + " ,l, " + chr$(45): rem  rec len = 45
EO  51050 rr = m: gosub 40000
JE  51060 print#1,left$(sp$,44)
NJ  51070 close 1: close 15: goto 51110
AK  51080 input " Name of data file[4 spcs]animals.dat
            [13 lefts] " ;f$
GA  51090 :
FL  51100 rem** teach first two animals
JK  51110 r1$ = " Does it live in the water " :rem  first
            question
HM  51120 r2$ = " Fish "  : rem  'yes' answer
FB  51130 r3$ = " Horse " : rem  'no' answer
HP  51140 z$ = chr$(0)
AF  51150 sp$ = "          [56 spaces]          "
GJ  51160 open 15,8,15
MN  51170 open 1,8,9,f$
KE  51180 rr = 1: gosub 40000
MF  51190 print#1,chr$(5);z$;left$(sp$,40)
IM  51200 gosub 40000: rr = rr + 1: gosub 40000
KD  51210 print#1,chr$(3);z$;chr$(4);z$;left$(r1$ + sp$,40)
MN  51220 gosub 40000: rr = rr + 1: gosub 40000
OM  51230 print#1,z$z$z$z$;left$(r2$ + sp$,40)
AP  51240 gosub 40000: rr = rr + 1: gosub 40000
DO  51250 print#1,z$z$z$z$;left$(r3$ + sp$,40)
DO  51260 close 15
GE  51270 end
```

BREAK KEY 64

Frank E. DiGioia
Athens, Georgia

. . .alter parameters while a program is running. . .



If you have ever used a mainframe computer, you are probably familiar with the BREAK key. If output is scrolling by too fast, you can halt it with the BREAK key and then cause it to resume, when you are ready, by pressing return. In addition to just stopping output, most mainframe computers allow one to give commands which affect program output while in break mode. These are usually referred to as BREAK-TIME COMMANDS. In this article we will examine a wedge which converts the RESTORE key on the C64 into a BREAK key, thus allowing the user to enter break-time commands to alter certain parameters while a program is running. This exciting capability can be a tremendous aid in program development and makes using other people's programs more enjoyable.

Why a BREAK Key?

Here's the scenario: You get a disk full of public domain software and boot up the first program (an address filer of course). You're instantly blinded by the authors choice of colors -- orange on cyan on purple. What do you do? Simply hit the BREAK key to interrupt the program, adjust the colors exactly the way you want them and then hit RETURN to continue execution. Easy! But that is only the beginning -- BREAK KEY 64 will also allow you to redirect output to or from the printer, set new default colors for RUN/STOP-RESTORE, make a hardcopy of the current text screen or perform a cold system reset. It is compatible with most utilities including Epyx Fastload, the DOS wedge, last issue's command wedge and function wedge and almost every non-commercial program there is. In addition, BREAK KEY 64 is open ended so you can easily add your own custom break-time commands.

How Does It Work?

Most readers are probably aware that when the RESTORE key is struck, it generates a Non-Maskable Interrupt (NMI) to the 6510 chip. That is, the 6510 finishes the instruction it is currently working on, saves the STATUS REGISTER and PROGRAM COUNTER on the stack and jumps to the address found at $0318/$0319. If the STOP key is not pressed, hitting the RESTORE key appears to do nothing even though it actually interrupts whatever the computer was doing and executes some ROM routines before returning from the interrupt. Clearly, all we need to do in order to trap the RESTORE key is to change the vector at $0318/$0319 to point at our own routine. We must be especially careful, however, not to change any registers or memory locations which could affect the BASIC or machine language program which will be interrupted by the break key. This means that care must be taken in selecting ROM routines to use in your break code since some of these routines use the floating point accumulator and other vital zero page locations as a workspace. One consequence of this limitation is seen in the very simple parser used by BREAK KEY 64. Rather than rely on ROM routines for numeric input, I opted to settle for simple one-character commands. It should be noted, however, that you could always copy all of zero page memory into a buffer then use any ROM routine you choose and finally replace the zero page memory before continuing program execution.

BREAK KEY 64 Description:

To enter break mode with BREAK KEY 64, whether a program is running or not, you need only to strike the RESTORE key. To leave break mode press RETURN. One thing which might confuse you at first is the fact that upon entering break mode, no message is printed to the screen to let you know that you are in break mode. This is to prevent the display from being corrupted in case you want to make a hardcopy of the screen. To enter a command, type a slash (/). The slash is not echoed until it is clear that you are not requesting a hardcopy of the screen. After you type the slash, the parser waits for a valid command key press. The following lines list the valid commands and explain what they do. The command key will always be echoed in the top right hand corner of your screen.

/H The H command makes a hardcopy of the screen. This command is perfect for copying down instruction screens, neat looking title screens, any low resolution graphics, etc. When the copy is complete, hit RETURN to continue the program. You can abort the hardcopy command by pressing the STOP key.

/P The P command redirects all subsequent output from the screen to the printer. This command simply opens a file to the printer using the same character set as the screen (ie upper/lower or upper/graphics) and then changes the default output device to device #4. Be certain that your printer is turned on before using /P.

/S This command 'undoes' the /P command. It closes the printer file and changes the default output device back to the screen. Note: If the /P command doesn't seem to work, it probably means that you never issued a /S after your last /P to clear BREAK KEY 64's internal output flag. Issuing a /S should correct the problem.

/C This command changes the current screen colors. After typing /C you are in COLOR mode. Press T to change the text color. Press E to change the Edge (border) color and and press G to change the (back)Ground color. When you have adjusted the colors to your liking, press RETURN to exit COLOR mode. You are then back to BREAK mode.

/D This command sets the current screen, border and text colors as the default colors for RUN–STOP/RESTORE. For instance, if you issue this command while the current screen colors are green, blue and gray and then change the colors to anything else, when you hit RUN–STOP/RESTORE the screen will go back to green, blue and gray.

/* The * command causes a cold system reset. This command is included in case the C64 ever refuses to obey commands from the keyboard. You should still be able to make it respond to the RESTORE key. This simple command is the same as SYS 64738.

You will undoubtedly think of some other commands to add to BREAK KEY 64 as you program and use other people's programs. Some ideas might include a HIRES screen dump or a BSOUT wedge to alter the print speed, etc.

Final Notes:

There are several minor limitations to BREAK KEY 64. (1) It should NEVER be used to interrupt disk operations such as LOAD, SAVE or printing a directory with the DOS wedge. Interrupting a disk operation will almost certainly cause the C64 to miss a handshaking signal with the drive causing the computer and drive to 'hang up'. Now, before you blame BREAK KEY 64, note that the computer will always get 'hung up' if you press the RESTORE key during a disk operation either with or without BREAK KEY 64 being present (if you don't believe this is true, type LOAD"$",8 and strike the RESTORE key a few times while the directory is loading). (2) BREAK KEY 64 doesn't allow recursion. That is, you can't reenter break mode until you finish the current break mode task. There is nothing wrong with allowing recursion if you want – – there just isn't much point to it since its main affect is simply that if you hit RESTORE ten times, you'll have to hit RETURN ten times to leave break mode. (3) It is possible to double or triple bounce the RESTORE key causing you to end up with the C64 default colors on the screen instead of the defaults you selected and BREAK KEY 64 will be deactivated. This situation does not normally occur unless you really try to make it happen. If it does occur, however, you will have to reactivate BREAK KEY 64 with SYS 50432.

There are two program listings included with this article. LISTING 1 is the BASIC loader for BREAK KEY 64. LISTING 2 is the assembly language source code for the program. There are several stand–alone subroutines included in BREAK KEY 64 which you may find useful in your own programs. These routines include LOPEN, LCLOSE, DECODE, STOP and CHGTXT.

BREAK KEY 64 is activated with:

SYS 50432

($C500) and is, of course, immune to RUN/STOP–RESTORE.

BREAK KEY 64: BASIC Loader

```
HO  100 rem basic loader for restore wedge
JC  110 rem by frank digioia  02/27/86
OK  120 rem sys 50432 to activate
GP  130 :
NN  140 for adr=50432 to 51007:read ml
HG  150 cs=cs+ml:poke adr,ml:next
HI  160 ifcs<>73014thenprint"error in data"
OB  170 :
KG  180 data 169,  18, 160, 197,  32,  30, 171, 169
BN  190 data  41, 141,  24,   3, 169, 197, 141,  25
HP  200 data   3,  96,  66,  82,  69,  65,  75,  32
ND  210 data  75,  69,  89,  32,  54,  52,  32,  73
KH  220 data  78,  83,  84,  65,  76,  76,  69,  68
AA  230 data   0,  72, 138,  72, 152,  72, 169, 223
IC  240 data 162, 198, 141,  24,   3, 142,  25,   3
DK  250 data  32, 239, 198, 208,   3,  76, 137, 197
DP  260 data  88,  32,  88, 197, 201,  13, 240,  10
JG  270 data 201,  47, 208, 245,  32,  94, 197,  76
IK  280 data  65, 197,  32,   7, 197,  76, 188, 254
EJ  290 data  32, 228, 255, 240, 251,  96,  32,  88
PJ  300 data 197, 201,  72, 240,  54, 201,  80, 240
FE  310 data  17, 201,  67, 240,  16, 201,  42, 240
JE  320 data  15, 201,  83, 240,  14, 201,  68, 240
MH  330 data  13,  96,  76, 127, 198,  76,  21, 198
FN  340 data  76, 226, 252,  76, 109, 198,  76,  67
IG  350 data 198,  32,  21, 253,  32, 163, 253,  32
CJ  360 data  24, 229,  32, 201, 198,  32,   7, 197
AG  370 data 108,   2, 160, 169, 146, 141, 231, 197
NG  380 data 169, 123,  32,   6, 199, 162,   0, 134
EJ  390 data 251, 169,   4, 133, 252, 160,   0, 177
EF  400 data 251,  32, 249, 197,  32, 232, 197,  32
GO  410 data 210, 255,  32, 239, 198, 240,  32, 200
NF  420 data 192,  40, 144, 235, 169,  13,  32, 210
BO  430 data 255, 173, 231, 197,  32, 210, 255, 165
```

```
DL  440 data 251,  24, 105,  40, 133, 251, 144,   2
AA  450 data 230, 252, 232, 224,  25, 144, 206,  32
AC  460 data 204, 255, 169, 123,  76,  43, 199, 146
LM  470 data 133, 215,  41,  63,   6, 215,  36, 215
CG  480 data  16,   2,   9, 128, 112,   2,   9,  64
NA  490 data  96,  72, 173, 231, 197,  48,  14, 104
FG  500 data  48,  10,  72, 169, 146, 141, 231, 197
II  510 data  32, 210, 255, 104,  96, 104,  16, 252
NB  520 data  72, 169,  18, 208, 240, 169,   3,  32
OF  530 data  95, 198,  32,  88, 197, 201,  13, 208
MD  540 data   1,  96, 201,  84, 208,   9, 238, 134
ND  550 data   2,  32, 147, 198,  76,  26, 198, 201
CL  560 data  69, 208,   6, 238,  32, 208,  76,  26
HJ  570 data 198, 201,  71, 208, 221, 238,  33, 208
BM  580 data  76,  26, 198, 169,   4,  32,  95, 198
OE  590 data 173, 134,   2, 141,  91, 198, 173,  32
MK  600 data 208, 141,  92, 198, 173,  33, 208, 141
OP  610 data  93, 198,  96,   1,   0,  10, 122, 141
PD  620 data  39,   4, 169,  47, 141,  38,   4, 169
FF  630 data  32, 141,  37,   4,  96, 169,  19,  32
JH  640 data  95, 198, 173,  94, 198,  16, 245, 169
OF  650 data 122, 141,  94, 198,  76,  43, 199, 169
NM  660 data  16,  32,  95, 198, 173,  94, 198,  48
EA  670 data 227, 169, 255, 141,  94, 198, 169, 122
EO  680 data  76,   6, 199, 169,   0, 133, 251, 169
JO  690 data 216, 133, 252, 169, 232, 133, 253, 169
NL  700 data 219, 133, 254, 160,   0, 173, 134,   2
CE  710 data 145, 251,  32, 178, 198,  32, 190, 198
MF  720 data 144, 243,  24, 169,   1, 101, 251, 133
ED  730 data 251, 144,   2, 230, 252,  96, 165, 252
AB  740 data 197, 254, 208,   4, 165, 251, 197, 253
MP  750 data  96, 173,  91, 198, 141, 134,   2,  32
NH  760 data 147, 198, 173,  92, 198, 141,  32, 208
MH  770 data 173,  93, 198, 141,  33, 208,  96,  72
IK  780 data 138,  72, 152,  72,  32, 239, 198, 208
LF  790 data   3,  76, 137, 197,  76, 188, 254, 141
PF  800 data   4, 199, 142,   5, 199,  32, 188, 246
FI  810 data  32, 225, 255,   8, 174,   5, 199, 173
EC  820 data   4, 199,  40,  96,   0,   0, 141,  60
AH  830 data 199, 173,  24, 208, 201,  21, 240,   3
JH  840 data 160,   7,  44, 160,   0, 173,  60, 199
DG  850 data 162,   4,  32, 186, 255, 169,   0,  32
LO  860 data 189, 255,  32, 192, 255, 174,  60, 199
PO  870 data  76, 201, 255,  72, 170,  32, 201, 255
OM  880 data 169,  13,  32, 210, 255,  32, 204, 255
EO  890 data 104,  76, 195, 255,  14,   0, 255, 255
```

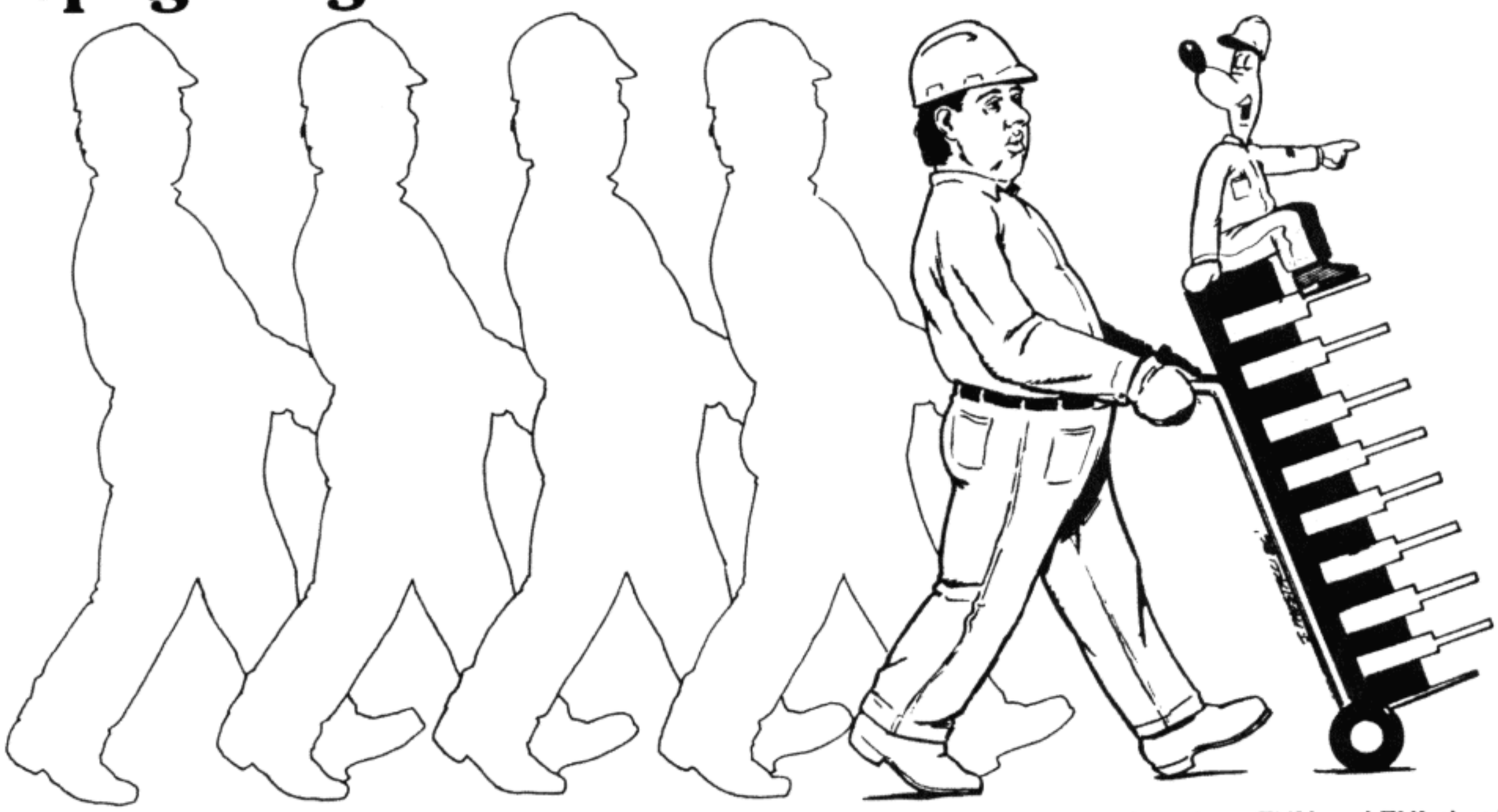
BREAK KEY 64: Source Code

```
BK  100 ;restore wedge -- break-time cmds.
LB  110 ;by frank e. digioia
FF  120 ;11/19/85
IP  130 ;
HI  140 *       =   $c500
MA  150 ;
FG  160 chrout  =   $ffd2     ;print char in .a
DN  170 getin   =   $ffe4     ;check keyboard
NC  180 cold    =   $fce2     ;cold system reset
ED  190 ;
OK  200         lda #<msg     ;point to mesg
 JJ 210         ldy #>msg
 LF 220         jsr $ab1e     ;print mesg
MF  230 ;
OF  240 brkint  lda #<rwdg    ;setup nmi wedge
FG  250         sta $0318     ;nmi vector
FB  260         lda #>rwdg
OM  270         sta $0319
EA  280         rts
NE  290 msg     .byte 'break key 64 installed'
MO  300         .byte $00
MK  310 ;
GB  320 rwdg    pha           ;save regs
OP  330         txa
AN  340         pha
FB  350         tya
EO  360         pha
 IO 370 ;
PB  380         lda #<brk2    ;disable recursion
 MI 390         ldx #>brk2
PE  400         sta $0318
GL  410         stx $0319
KB  420 ;
OD  430         jsr stop      ;scan stop key
PO  440         bne *+5       ;not pressed
NF  450         jmp newrsr    ;run/stop-restore
FM  460         cli           ;allow keyboard
ME  470 ;
HM  480 wrloop  jsr waitlp    ;check keyboard
 JA 490         cmp #$0d      ;" carriage return?
 JB 500         beq return
OC  510         cmp #"/"      ;" command coming?
GL  520         bne wrloop    ;no/keep waiting
 OI 530         jsr chkcmd    ;check command
 JB 540         jmp wrloop    ;wait for more
MJ  550 ;
PE  560 return  jsr brkint    ;restore wedge
GH  570         jmp $febc     ;restore regs/rti
KL  580 ;
 IC 590 waitlp  jsr getin     ;check keyboard
 EJ 600         beq waitlp    ;" key pressed?
OE  610         rts
CO  620 ;
CC  630 chkcmd  jsr waitlp    ;get cmmd byte
NO  640         cmp #"h"      ;" hardcopy?
HO  650         beq hrdcpy
 MJ 660         cmp #"p"      ;" printer?
OB  670         beq jmptab
MP  680         cmp #"c"      ;" colors?
MP  690         beq jmptab+3
ED  700         cmp #"*"      ;" reset?
MB  710         beq jmptab+6
PM  720         cmp #"s"      ;" restore screen?
MD  730         beq jmptab+9
 JF 740         cmp #"d"      ;" defaults?
CG  750         beq jmptab+12
AA  760         rts           ;go to wait loop
 IH 770 ;
FB  780 jmptab  =   *         ;jump table
 IO 790         jmp prnter
DL  800         jmp colors
GE  820         jmp rscrn
FE  830         jmp defalt
OL  840 ;
BH  850 newrsr  =   *         ;new stop-restore
AD  860         jsr $fd15     ;initialize vectors
OG  870         jsr $fda3     ;init sid/vic regs
LE  880         jsr $e518     ;reset screen
EB  890         jsr setdef    ;set default colors
 GI 900         jsr brkint    ;initialize wedge
 BA 910         jmp ($a002)   ;basic warm start
OA  920 ;
HH  930 ;hrdcpy -- dumps the text screen
BF  940 ;to the printer. use to save
GB  950 ;lo-res graphics, notes, etc.
GD  960 ;
NA  970 open    =   $ffc0     ;open a file
HC  980 close   =   $ffc3     ;close a file
 LO 990 clrch   =   $ffcc     ;set standard i/o
NN  1000 chkout =   $ffc9     ;set output device
MG  1010 setlfs =   $ffba     ;set file parms
 IA 1020 setnam =   $ffbd     ;set fn parms
MH  1030 ;
 AG 1040 hrdcpy =   *         ;dump screen to printer
GB  1050        lda #$92      ;normal mode flag
 AL 1060        sta revflg    ;set reverse flag
 JJ 1070        lda #$7b      ;file #123
FK  1080        jsr lopen     ;open printer file
BM  1090        ldx #$00
 OD 1100        stx $fb       ;store screen adr
 JI 1110        lda #$04
FC  1120        sta $fc       ;as a pointer
AO  1130 ;
 AI 1140 rwloop =   *         ;loop over rows
 OG 1150        ldy #$00      ;reset column count
 OP 1160 ;
 BJ 1170 chloop =   *         ;loop over columns
```

```
MN 1180          lda  ($fb),y   ;get chr from screen
DE 1190          jsr  chkmde    ;" normal or reverse?
KA 1200          jsr  decode    ;screen code to ascii
NN 1210          jsr  chrout    ;print the char.
EF 1220          jsr  stop      ;scan stop key
KL 1230          beq  hend      ;yes/finish up
IL 1240          iny
BF 1250          cpy  #$28      ;" 40th column yet?
HC 1260          bcc  chloop
MG 1270 ;
LC 1280 eopl     =    *         ;end of physical line
NG 1290          lda  #$0d      ;carriage return
LP 1300          jsr  chrout    ;output it
LK 1310          lda  revflg    ;get last mode value
GC 1320          jsr  chrout    ;set the mode
KP 1330          lda  $fb       ;update screen pointer
CL 1340          clc
PM 1350          adc  #$28      ;to next row
MG 1360          sta  $fb
OK 1370          bcc  *+4       ;" need to incr hi byte?
FG 1380          inc  $fc
ID 1390          inx             ;incr row counter
GJ 1400          cpx  #$19      ;" 25th row yet?
KO 1410          bcc  rwloop
CA 1420 ;
PK 1430 hend     =    *         ;finished!
OH 1440          jsr  clrch     ;restore normal i/o
KC 1450          lda  #$7b
FE 1460          jmp  lclose    ;close printer
ED 1470 ;
OC 1480 revflg   .byte $92
IE 1490 ;
IE 1500 ;decode -- this routine converts
KE 1510 ;a character from its screen
HE 1520 ;code representation to ascii.
KE 1530 ;this is copied directly from
GF 1540 ;the kernal rom at $e63e (cursor
LB 1550 ;stuff removed).
OI 1560 ;
NO 1570 decode   =    *         ;screen code to ascii
FH 1580          sta  $d7       ;store code
CJ 1590          and  #$3f      ;mask out top 2 bits
PK 1600          asl  $d7       ;mult code by two
LC 1610          bit  $d7       ;condition flags
IK 1620          bpl  *+4       ;" bit 7 clear?
CL 1630          ora  #$80      ;no - set bit 7 in .a
NM 1640          bvs  *+4       ;" bit 6 set?
JP 1650          ora  #$40      ;no - set bit 6
BH 1660          rts             ;ascii code now in .a
MP 1670 ;
AP 1680 ;chkmde -- checks on normal or
KF 1690 ;reverse & sends code to printer
KB 1700 ;
OH 1710 chkmde   pha             ;save screen code
CP 1720          lda  revflg    ;" what mode current?
FM 1730          bmi  nermal    ;normal mode/skip
EM 1740          pla             ;retrieve screen code
LO 1750          bmi  chkx      ;reverse/no sweat
OC 1760          pha             ;save code again
JE 1770          lda  #$92      ;normal/kill reverse
ID 1780 hc1      sta  revflg    ;toggle flag
PE 1790          jsr  chrout    ;print the code
MI 1800          pla             ;restore screen code
AO 1810 chkx     rts
CJ 1820 ;
GG 1830 nermal   pla             ;get screen code
CB 1840          bpl  chkx      ;normal/no sweat
II 1850          pha             ;save code again
AI 1860          lda  #$12      ;rev/switch mode
FC 1870          bne  hc1       ;finish up
OM 1880 ;
ED 1890 ;/c command (colors)
CO 1900 ;
CE 1910 colors   lda  #$03      ;code for 'c'
PB 1920          jsr  output    ;print command name
GB 1930 zloop    jsr  waitlp    ;wait for key
AE 1940          cmp  #$0d      ;" return?
JG 1950          bne  *+3       ;no/keep going
EJ 1960          rts
IC 1970 ;
AK 1980          cmp  #"t"      ;" text?
CO 1990          bne  *+11      ;no/check on edge
CN 2000          inc  $0286     ;inc text color
HP 2010          jsr  chgtxt    ;change text color
KA 2020          jmp  zloop
EG 2030 ;
GE 2040          cmp  #"e"      ;" edge (border)?
MO 2050          bne  *+8       ;no/check on ground
CN 2060          inc  $d020     ;inc border color
MD 2070          jmp  zloop
GJ 2080 ;
FK 2090          cmp  #"g"      ;" (back)ground?
MC 2100          bne  zloop     ;no/must be typo
OK 2110          inc  $d021     ;inc bckgrnd color
OG 2120          jmp  zloop
IM 2130 ;
JB 2140 ;/d command (set defaults)
MN 2150 ;
LE 2160 defalt   =    *
OF 2170          lda  #$04      ;code for 'd'
BJ 2180          jsr  output    ;print it
DM 2190          lda  $0286     ;get text color
GN 2200          sta  d1        ;save it
PE 2210          lda  $d020     ;get edge color
MO 2220          sta  d2        ;save it
GI 2230          lda  $d021     ;get ground color
CA 2240          sta  d3        ;save it
BO 2250          rts             ;that's all folks
KE 2260 ;
MM 2270 d1       .byte $01
GN 2280 d2       .byte $00
GB 2290 d3       .byte $0a
IB 2300 pflag    .byte $7a
MH 2310 ;
GA 2320 ;output subroutine
AJ 2330 ;
LK 2340 output   =    *         ;echo command
CL 2350          sta  $0427     ;corner of screen
ID 2360          lda  #$2f      ;slash code
FM 2370          sta  $0426     ;corner of screen
KO 2380          lda  #$20      ;space code
IN 2390          sta  $0425     ;corner of screen
EP 2400 oxit     rts             ;all done!
AO 2410 ;
EH 2420 ;/s command (output to screen)
EP 2430 ;
FO 2440 rscrn    =    *
BL 2450          lda  #$13      ;code for 's'
AM 2460          jsr  output    ;echo command
MH 2470          lda  pflag     ;" printer online?
GA 2480          bpl  oxit      ;no/don't bother
AC 2490          lda  #$7a      ;file #122
MJ 2500          sta  pflag     ;set flag/offline
GH 2510          jmp  lclose    ;close it up
OE 2520 ;
ED 2530 ;/p command (output to printer)
CG 2540 ;
MA 2550 prnter   =    *
HA 2560          lda  #$10      ;code for 'p'
OC 2570          jsr  output    ;echo command
KO 2580          lda  pflag     ;" printer online?
AE 2590          bmi  oxit      ;yes/don't bother
OB 2600          lda  #$ff      ;set printer online
FH 2610          sta  pflag     ;set flag
CK 2620          lda  #$7a      ;file #122
LI 2630          jmp  lopen
GM 2640 ;
MD 2650 ;text color subroutines
KN 2660 ;
HM 2670 colrst   =    $d800     ;start color mem
CJ 2680 colrnd   =    $dbe8     ;end color mem + 1
IP 2690 ;
BC 2700 chgtxt   lda  #<colrst  ;set up adr
ND 2710          sta  $fb       ;pointer for start
JL 2720          lda  #>colrst  ;and end of color
CP 2730          sta  $fc       ;memory for fill.
MG 2740          lda  #<colrnd
AO 2750          sta  $fd
MH 2760          lda  #>colrnd
HP 2770          sta  $fe
CF 2780 ;
OG 2790          ldy  #$00      ;.y is always zero
GG 2800 ;
AN 2810 cfloop   lda  $0286     ;get text color
EL 2820          sta  ($fb),y   ;fill color memory
JB 2830          jsr  incrmt    ;increment $fb/$fc
NK 2840          jsr  chek      ;compare start/end
FP 2850          bcc  cfloop    ;" end reached yet?
CK 2860 ;
FK 2870 incrmt   clc             ;16 bit increment
LO 2880          lda  #$01      ;increment of 1
AH 2890          adc  $fb       ;increment lo byte
KC 2900          sta  $fb       ;update it
FP 2910          bcc  *+4       ;" increment hi byte?
JG 2920          inc  $fc
OF 2930          rts
CP 2940 ;
PN 2950 chek     lda  $fc       ;compare 16 bit #'s
HJ 2960          cmp  $fe       ;compare hi bytes
PJ 2970          bne  *+6       ;'nuff said/return
MD 2980          lda  $fb       ;hi bytes equal so
PL 2990          cmp  $fd       ;low bytes get to
EJ 3000          rts             ;make the decision
ID 3010 ;
IF 3020 setdef   lda  d1        ;default text col
BA 3030          sta  $0286     ;set text color
BF 3040          jsr  chgtxt    ;fill color mem
NF 3050          lda  d2        ;default edge color
DO 3060          sta  $d020     ;set border color
LO 3070          lda  d3        ;default ground color
IM 3080          sta  $d021     ;set background col
OP 3090          rts
CJ 3100 ;
MM 3110 brk2     =    *         ;user break 2
GA 3120          pha             ;save regs
OO 3130          txa
AM 3140          pha
FA 3150          tya
EN 3160          pha
IN 3170 ;
MP 3180          jsr  stop      ;scan stop key
AI 3190          bne  *+5       ;not pressed/continue
ON 3200          jmp  newrsr    ;pressed/warm reset
KK 3210          jmp  $febc     ;restore regs & rti
KA 3220 ;
JE 3230 stop     sta  areg      ;save .a in memory
GH 3240          stx  xreg      ;save .x in memory
MO 3250          jsr  $f6bc     ;set stop flag
AG 3260          jsr  $ffe1     ;read stop flag
BO 3270          php             ;save status word
LJ 3280          ldx  xreg      ;get .x back
GN 3290          lda  areg      ;get .a back
FC 3300          plp             ;get status word
KN 3310          rts
OG 3320 ;
FJ 3330 areg     .byte $00
NM 3340 xreg     .byte $00
MI 3350 ;
EM 3360 ;lopen -- this routine opens a
LB 3370 ;file to the printer (file no.
DI 3380 ;is in .a on entry). the
MM 3390 ;printer is opened to whichever
CO 3400 ;mode the screen is in. i.e. if
DD 3410 ;the screen is in the uppercase/
FB 3420 ;lowercase mode, the printer
AM 3430 ;will be also. the same applies to
BE 3440 ;uppercase/graphics mode.
AP 3450 ;
FK 3460 lopen    =    *         ;open a printer file
KM 3470          sta  lfn
KE 3480          lda  $d018     ;which char setprint
HL 3490          cmp  #21       ;uppercase/graphics
GC 3500          beq  ug
NO 3510          ldy  #$07      ;sa for printer (u/l)
BA 3520          .byte $2c      ;skip next instr
MI 3530 ug       ldy  #$00      ;sa for upper/graphic
CN 3540          lda  lfn
EK 3550          ldx  #4        ;# for serial printer
CL 3560          jsr  setlfs
FB 3570          lda  #$00
NL 3580          jsr  setnam
PA 3590          jsr  open      ;system open routine
PD 3600          ldx  lfn       ;get file number
BD 3610          jmp  chkout    ;open output channel
KJ 3620 ;
EK 3630 ;
NC 3640 ;lclose - routine to close a
JG 3650 ;file to the printer. filenum
HK 3660 ;in .a upon entry.
MM 3670 ;
HI 3680 lclose   =    *         ;close a printer file
LD 3690          pha             ;save file number
NH 3700          tax             ;get file number in .x
FP 3710          jsr  chkout    ;open the channel
LO 3720          lda  #$0d      ;carriage return
PF 3730          jsr  chrout    ;send to printer
KK 3740          jsr  clrch     ;reset standard i/o
NJ 3750          pla             ;get file number
PE 3760          jmp  close     ;close the file
AD 3770 ;
KF 3780 lfn      .byte $0e
EE 3790 ;
EL 3800 .end
```

MOVE:
A General Purpose Propagating Move Routine

R. J. de Graff
Winnipeg, Manitoba



As documented in a previous issue (July 85 Bits & Pieces), the built in memory move routine can be used to move blocks of memory hither and yon. Unfortunately, the page 0 locations that must be primed are also used by BASIC. Therefore, to use this routine from within a BASIC program requires that a front end be built to copy the parameters from user selected addresses to those used by the move routine.

Rather than do this I decided to write my own move routine, MOVMEM. This routine resides in the ever popular cassette buffer and is a mere 37 bytes long. Its parameters are poked into page 0 locations 25–30. The calling program specifies source address start, destination address start and the number of bytes to copy (all in low byte/high byte format).

MOVE is fast. It also has the advantage of being able to perform what is known as a propagating move. More on that in a moment. For now, let's look at how MOVE can be used from BASIC. One example that comes to mind is moving BASIC from ROM into the underlying RAM so that it can be modified. Typically this is done by the loop:

```
100 for i = 40960 TO 49151
105   poke i,peek(i)
110 next i
115 poke 1,54
```

Timing this loop we find it takes 37 seconds to execute. Now let's see how the same thing is done using MOVE. For the sake of clarity I will define two functions FNH and FNL that break up a 16 bit integer into high and low bytes suitable for poking. The parameters for MOVE are as follows:

BYTES 25–26	source address (low/high)
BYTES 27–28	destination address (low/high)
BYTES 29–30	number of bytes (low/high)

```
100 def fnh(x) = int(x/256)
105 def fnl(x) = x - 256 * fnh(x)
110 :
115 bs = 40960 : rem basic start
120 nb = 8192  : rem length of basic
125 :
130 poke 25, fnl(bs) : poke 26, fnh(bs)
135 poke 27, fnl(bs) : poke 28, fnh(bs)
140 poke 29, fnl(nb) : poke 30, fnh(nb)
145 sys 828
```

While this requires more code it executes in less than one second. If the move is to be done frequently (copying bit maps in hi-res mode for example) then the grunt work can be moved into a subroutine such as the following:

```
900 poke 25, fnl(sa): poke 26, fnh(sa) :rem source address
905 poke 27, fnl(da): poke 28, fnh(da) :rem destination addr
910 poke 29, fnl(nb): poke 30, fnh(nb) :rem number of bytes
915 sys 828 : return
```

Just don't forget to define FNH and FNL.

As I mentioned earlier, this routine can be used to perform a propagating move. This type of move results when the destination address is one more than the source address. For example, let's say we want to clear out an area of memory 1000 bytes long to use for a hi-res bit map. Assuming that BM contains the address of our bit map, this is accomplished by the following:

```
200 poke bm,0
205 sa = bm: da = bm + 1: nb = 999: gosub 900
```

Fast and painless. Exactly what has happened? The bytes were copied as follows:

SA + 0 copied to SA + 1 (which is DA)
SA + 1 copied to SA + 2 (DA + 1)
SA + 2 copied to SA + 3
.
.
.
SA + 999 copied to SA + 1000

The zero value poked in line 200 was propagated all through the 1000 bytes.

This is a side effect not possible with the built in routine due to the algorithm used. The built in routine copies blocks in the order byte 0 then bytes 255 to 1 whereas MOVE copies bytes in the order 0, 1, 2, . . . 255. Assembler source and BASIC loader follow.

Program Listing (Commodore Assembler)

```
*       =   828
;
srce    =   25          ;parameter - location of srce address
dest    =   27          ;parameter - location of dest address
nbytes  =   29          ;parameter - location of number of bytes
;
nblks   =   nbytes+1    ;number of full blocks to move
;
move    lda  nblks      ;get number of full blocks to move
        beq  donebl     ;if none then test for partial block
        lda  #0         ;set comparison value for transfer
start   sta  remain     ;256 byte blocks (0=256)
        ldy  #0         ;set index to point to first byte of block
;
nxtbyt  lda  (srce),y   ;get next source byte
        sta  (dest),y   ;store at next destination slot
        iny             ;set index to point to next byte
        cpy  remain     ;compare to remaining bytes value
        bne  nxtbyt     ;if not all moved then do next byte
;
        inc  srce+1     ;increment srce address by one block
        inc  dest+1     ;increment dest address by one block
        dec  nblks      ;decrement number of full blocks to move
        bmi  done       ;if minus then we are done all bytes
        bne  nxtbyt     ;if positive we have more blocks to do
;
donebl  lda  nbytes     ;do we have a partial block
        bne  start      ;if yes then move it
done    rts             ;return to calling routine
;
remain  *=*+1           ;number of bytes to move in current block
```

BASIC Loader Subroutine

```
900 cs = 0
905 for i = 828 to 864
910   read b : poke i,b : cs = cs + b
915 next i
920 :
925 if cs <> 4327 then print " checksum error " : stop
930 :
935 data 165,  30, 240,  27, 169,   0, 141
940 data  96,   3, 160,   0, 177,  25, 145
945 data  27, 200, 204,  96,   3, 208, 246
950 data 230,  26, 230,  28, 198,  30,  48
955 data   6, 208, 236, 165,  29, 208, 227
960 data  96
965 :
970 return
```

Editor's Note

MOVE works great for moving all sorts of stuff; sprite definitions, colour memory, low-res screen data, or even snapshots of zero-page and the like. And, as mentioned in News BRK, feel free to use any subroutine published in The T. in any program, anytime, anywhere, and for any purpose. M.Ed.

Bit Addressing Of Sprite Controls

Stacy McInnis
Upland, California

One good reason to switch from BASIC to assembly language programming is to increase the speed with which you can move and change your sprites. An added advantage is the ease with which you can do bit manipulations. Several of the locations that control sprites must be addressed at the bit level. Among these are $D010 which contains the most significant bits of the X positions of your sprites on the screen and $D015 which controls the enabling and disabling of individual sprites. Each byte that contains sprite information for all the sprites is arranged in the same order. Sprite zero is represented by the least significant or right most bit while sprite seven is represented by the most significant or left most bit. As there are eight sprites allowed and eight bits in a byte, this works out very nicely.

Looking at the 6510 microprocessor instruction set, it is clear that there are no instructions that will let you address any selected bit with a single instruction. However, a combination of these instructions can be used to do the job. But first, let us define some data. Find sixteen locations that you will not be disturbing and dedicate them to holding masks. A mask is a number that lets you use masking tape to prevent painting beyond your desired area. Our mask will be set up to allow the changing or testing of a single bit without disrupting the other bits in a byte.

Example Of Defining Mask

```
; The first mask array will be used with the ORA instruction to turn
a bit on
;
maskor =    *
        .byt $01        ;bits 00000001 sprite 0
        .byt $02        ;bits 00000010 sprite 1
        .byt $04        ;bits 00000100 sprite 2
        .byt $08        ;bits 00001000 sprite 3
        .byt $10        ;bits 00010000 sprite 4
        .byt $20        ;bits 00100000 sprite 5
        .byt $40        ;bits 01000000 sprite 6
        .byt $80        ;bits 10000000 sprite 7

; The second mask array will be used with the AND instruction to
turn a bit off
;
;
maskan = *
        .byt $fe        ;bits 11111110 sprite 0
        .byt $fd        ;bits 11111101 sprite 1
        .byt $fb        ;bits 11111011 sprite 2
        .byt $f7        ;bits 11110111 sprite 3
        .byt $ef        ;bits 11101111 sprite 4
        .byt $df        ;bits 11011111 sprite 5
        .byt $bf        ;bits 10111111 sprite 6
        .byt $7f        ;bits 01111111 sprite 7
```

Now, let us try the simple task of enabling Sprite 2. To enable sprite 3, set bit 3 of location $d015 to 1.

```
ldx  #3          ;x contains a 3
lda  $d015       ;the sprite enable register
and  maskan,x    ;use $f7 so no bits are altered except bit
                 3
sta  $d015       ;reset the register and enable sprite 3
```

But why write all that code with the masks and indexes? If you want to enable sprite 3, why not just write:

```
lda  $d015
ora  #$08
sta  $d015
```

Certainly this would be the better way to enable a single sprite. But in most instances the same code is being executed for several sprites and so it is desirable to use indexing.

Now for a complex example. Assume you would like to expand multi–colored sprites horizontally. You also want to disable high–resolution sprites. But you only want to do this for sprites 2 to 7. Sprites 0 and 1 are to remain untouched.

```
        lda  #5          ;6 sprites total
loop    =    *
        lda  $d01c       ;the multi color register
        ora  maskor+2,x  ;select only bit to consider
        beq  maskor      ;zero is to be disabled
;
; if here is multi–color so expand it
;
        ora  $d01d       ;horizontal expansion
        sta  $d01d       ;reset expansion
        bne  eloop       ;always branch
;
; are here if have a single colored sprite to be disabled
;
dable   =    *
        lda  maskan+2,x  ;select mask
        and  $d015       ;the enable/disable register
        sta  $d015       ;reset register with sprite disabled
;
eloop   =    *
        dex              ;index to next mask
        bpl  loop        ;loop until sprites 7 to 2 are tested
```

Another instruction that works well with mask is the EOR. Assume that you wish to horizontally expand a sprite if it is unexpanded or unexpand if it is expanded. In other words, whatever the bit is, we want to change it. Again, the sprite to be changed is in register X.

```
lda  $d01d       ;the horizontal expansion register
eor  maskor,x    ;change the bit
sta  $d01d       ;reset register
```

If you are writing software that is sprite intensive, your masks will be well worth the sixteen bytes you took to define them.

Sprite Numbers

James A. Lisowski
S. Milwaukee, Wisconsin

A painless way to flash numbers on text or graphics screens.

The sprite number technique is potentially useful for just about any program that needs a numerical display. Normally, sprites are made into graphic shapes for use as game markers or other indicators and require some drawing to define the sprite's shape. The demonstration program presented here will take a three digit positive number and create a sprite that displays the same value – without any drawing on your part. What's so great about a sprite number? Well, first of all the sprite can be a colour that is different than the existing foreground and background. A sprite can also appear in front or behind an existing text or graphic display, including a high resolution screen – without altering it. Of course, the sprite number can also move anywhere around the screen as fast or slow as desired. The sprite number could also be flashed on and off or in different colors to capture attention. And, best of all, the sprite number can be expanded to twice the size of normal text characters.

If those features don't give you any application ideas, here are a few suggestions: Sprite numbers make great action game scorecards. For example, when that big ship gets hit by your laser, a sprite number can be placed right next to the hit to show the bonus points scored. With extra programming, you could make the number fade in, break up or flash. If you need an X / Y position display for that hi–res drawing program, sprite numbers will do the job without changing your drawing or taking up space. Likewise, a digital clock made of sprites won't disrupt your wordprocessing. If your application includes process control, an out of range sensor reading can come into view at any time. If you want to get fancy, the sprite can be part number and part graphic – say a digital display and a bar chart representation of the same value, for ease of viewing and precision. I'm sure you can think of many more uses for sprite numbers. Although my program does have REMarks, I'll explain the program, line by line, so that anyone who needs sprite numbers can understand how they are made and used.

All of the lines before line 500 just do some initial set up work; a sample mainline program covers lines 500 through 760 and the actual sprite number subroutine occupies lines 900 through 1270. The example uses sprite number zero for number displays, of course, any sprite will work just as well. Here is how the demonstration program works:

Line 30 makes variable VS equal to the address of the VIC II graphic chip's first register so that any register may be selected by adding an offset value VS (eg. VS+16 selects register 16). Lines 30 and 40 create variables (SX, SY, SN), that correspond to sprite zero's horizontal and vertical position control registers as well as the register that controls whether the sprite is visible or not, the enable register. These variables are used often enough to merit special names. Line 60 tells VIC II to look for the shape information for sprite zero in the 13th area reserved for sprite shapes. (In the normal VIC II configuration, area 13 runs from decimal location 832 to 895.) If this number is changed to another value and the corresponding shape area is filled with a number shape, one sprite can be used to display several predefined sprite numbers, each one switched in by changing the shape area number in location 2040. Line 90 expands the size of sprite 0 in the X and Y dimensions for emphasis. If this line were not present (or if the one values were changed to zeros), the sprite numbers would be the same size as the normal characters, but when they are expanded to twice the normal size, they really stand out. With various combinations of one and zero, the sprite numbers can also have other height / width ratios.

OK, so far the program has done some fairly normal sprite operations. Now comes the tricky part: stealing the dot patterns that make up a number shape from the 64's character ROM. In most computers, the character ROMs can only be read by the computer's video circuitry. If you wanted to use or alter the character shapes, you would have to draw the patterns and then duplicate them as one and zero (video dot on and off) patterns for your program's use. This is not only tedious but it also uses much of BASIC program memory just to store character patterns. The Commodore 64 has two very nice features that can be used to overcome these problems. One feature is that the microprocessor and the VIC II video chip can directly read the contents of the character ROMs; the other feature is plenty of RAM memory to store user defined character sets without using up BASIC program space. Here's how my program takes advantage of these attributes:

Lines 130 and 142 turn the usual timekeeping interrupt OFF for a while and switch the character ROM into the microprocessor's normal address space so that the number patterns can be read directly. (If the interrupt routine is not switched off, the computer will "freeze up" and not respond to normal commands because the routine doesn't allow for the character ROM's presence.) A side effect is that, since the interrupt routine normally updates the system real–time clock, (the time value found in variable TI$), this time value will not be correct and will "lose time" as long as the interrupt routine is turned off. When switched in, the character ROM appears at location

53248. Since each pattern takes up eight bytes and the number patterns start at the 48th character position, Line 160 sets variable CS to the starting address of the ROM's number patterns. Line 180 sets variable CM to a value that corresponds to a section of high RAM memory that BASIC never uses. Line 210 reads 80 bytes (the 10 number patterns) from ROM and stores them in the high RAM area for later use. Lines 230 to 240 finish the procedure by switching the ROM out and turning the interrupt routine back on. Line 310 blanks out any old data in the the sprite shape memory area to complete the set up section.

The demonstration mainline starts on line 530. This is a short program that shows how to use the sprite number subroutine. In normal use, you would keep the set up and subroutine sections and create your own application mainline. Line 530 makes sure that the MSB (Most Significant Bit) of the sprite's X position is off, so that the sprite travels on the left side of the video screen. Line 560 sets the sprite's X/Y initial screen position. Line 590 clears the screen and prints a title in normal text characters. Lines 620 and 650 generates random colors and three digit number values (NU) for the sprite. Line 700 calls the actual sprite number subroutine. Lines 740 and 745 move the finished sprite number horizontally across the screen to show off a bit, then disables (turns off) the sprite, resets the X screen position and goes back to the random number / color section to start over with a new sprite. Occasionally you won't see the sprite because it is the same color as the background. If you wish you can change the mainline to make the sprite move in the Y direction or over some text background.

Finally, here is what the sprite number subroutine does: Line 1000 disables (turns off) the sprite so that the transition from one number pattern to the next is invisible. (If you want to watch the process change the zero in Line 1000 to a one.) Line 1030 blanks out the old sprite pattern. Line 1070 formats the number (NU) to become a sprite. First the number is changed into string format with the STR$ function. Since the random number string's length can range from two characters (a space, because it is a positive number, and one digit), to four characters (a space and three digits), a space is added to the right side of the number string and the right–most three characters of this total combination are extracted with the RIGHT$(,3) statement. This operation will always yield a three character string, with leading spaces and place it into variable NU$. Now that the number string has been put together properly, the next operation is to take it apart again!

Line 1130 starts a FOR / NEXT loop that selects the characters, one by one. Also a counter, IN, that will be used later is reset to zero. The MID$ function in Line 1140 extracts one character from NU$ and places it into CH$. The first time through the loop, the left–most character is selected, then the middle character and then the right–most character, according to the position value (CH + 1). To save time, since the sprite has been cleared to a blank state, if the extracted character is a space, no further processing is done and Line 1150 directs program execution to the NEXT statement in Line 1230. Otherwise, the extracted digit character is changed back to number form with the VAL function and placed in ND. ND will be used as an index to the proper starting position of that digit's character pattern.

The FOR loop in Line 1200 steps the variable I through eight values that correspond to eight bytes in the sprite's shape memory. Sprite shape memory is linear, but the resulting sprite is displayed in a format where three pattern bytes lie side by side. The STEP index of 3 in Line 1200 places the eight dot pattern bytes so that they all line up in the same vertical column. Also in Line 1200 the variable CH (which changes value from zero to one as the FOR loop in Line 1130 progresses) offsets the sprite shape memory value so that the first character pattern goes into the first sprite display column, the second digit pattern goes into the middle column and the right digit pattern goes in the right–most sprite column. This is definitely not your average FOR / NEXT loop but it does work! Line 1210 takes this sprite shape memory position (I) and fills it with a byte of the proper character pattern from the high RAM storage area. To figure out which byte to take (PEEK), the start of high RAM number pattern storage (CM) is added to the starting position of the desired digit (ND*8, the NDth digit position times eight pattern bytes per digit) plus the index IN which steps through (IN = IN + 1), the eight sequential pattern bytes that make up the digit shape. Thus the result of Lines 1200 through 1220 are that the desired digit pattern is taken out of high RAM and placed in the proper position in the sprite. The NEXT statement in Line 1230 directs program execution back to the FOR statement in Line 1130 which then selects the next digit character to be loaded into the sprite (and resets index IN), until all three string characters have been processed.

At this point the sprite number has been built, but the sprite is still invisible. Line 1270 enables the sprite and RETURNs execution back to the mainline program. As outlined above, the mainline merrily moves the new sprite along then goes back and creates a new random sprite number. If you've followed all of these math and logic brain twisters you must now think that computers are pretty wonderful and that programmers ARE CRAZY! However, if you are still feeling like an experiment or two might clarify the operation of this demonstration, I would suggest:

1) changing Line 1000 to: 1000 POKE SN,1 (so that the sprite is enabled and you can see what is happening) and
2) sprinkle in a few PRINT statements that will show the values of those obscure variables.

When you are done with that, here are a few other things to ponder. First of all, if you want, you can dispense with copying the character patterns to high RAM and just take them from the character ROM each time they are needed. But remember, the real–time clock loses time each time you disable the interrupt and if anyone ever hits RUN / STOP while the program is gathering patterns, the next button to press is ON/OFF (or RESET if you have it).

Another note is that this routine does not use the entire sprite. There is still room in the bottom portion of the sprite to place a second set of three digits. (If you want to see this area change

the zero in line 310 to 255.) If you want to add the second number to the sprite just call the subroutine twice and add an offset to I in Line 1210 to place the second pattern a little lower in the sprite. Used this way, a dual number sprite makes a useful X/Y position readout for your hirez drawing program or a two player scorecard for your favorite action game. The only limitation to two-in-one sprite numbers versus two separate sprite numbers is that both numbers have the same color in the former case but can be different colors in the latter.

One other drawback for fast changing programs, is that all the math makes for a slow routine. Compiling and integer variables will help but the fastest routine would be pure machine language. And that, as they say in computer class, is left for the reader as an exercise.

The Sprite Numbers program with demo

```
JL  1 rem sprite number demo v4
NK  2 rem james a. lisowksi, 902 willow ln.
DN  3 rem s. milwaukee, wi 53172
BO  4 rem public domain software
JH  5 :
PK  10 rem----set up sprite zero
II  20 :
II  30 vs = 53248:rem first vic-ii register
KF  40 sx = vs:sy = vs + 1:rem sprite x/y reg.
EB  50 sn = vs + 21:rem sprite enable reg.
MI  60 poke2040,13:rem sprite mem area 13
DM  79 :
LK  80 rem remove rem in line 90 for
AO  82 rem un-expanded character size
IJ  90 pokevs + 29,1:pokevs + 23,1
IN  100 :
OK  110 rem----set up number patterns
MO  120 :
MC  130 rem turn interupt off, switch rom in
KP  140 poke56334,peek(56334)and254
OL  142 poke1,peek(1)and251
FA  145 :
AP  150 rem start of rom number patterns
AH  160 cs = 53248 + 8*48
AI  170 rem start of ram storage area
IC  180 cm = 49152
NC  190 rem take character info from rom
FJ  200 rem and place in into protected ram
EC  210 fori = 0to80:pokecm + i,peek(cs + i):next
DA  220 rem turn interupt on, switch rom out
IL  230 poke1,peek(1)or4
OO  240 poke56334,peek(56334)or1
OG  250 :
BD  300 rem----clear sprite pattern
IH  310 fori = 832to894:pokei,0:next
EL  320 :
OL  330 :
BF  500 rem = = = = demo mainline = = = =
CH  510 :
NF  520 rem set msb of x position to off
MB  530 pokevs + 16,0
AJ  540 :
JP  550 rem set sprite x/y position to 100
ID  560 pokesx,100:pokesy,100
OK  570 :
HN  580 rem clear and title the screen
HF  590 print " Ssprite number demo 1234567890 "
MM  600 :
PL  610 rem set sprite random sprite color
PD  620 pokevs + 39,rnd(0)*15
KO  630 :
KG  640 rem get a 1 to 3 digit positive #
DF  650 nu = int(rnd(0)*999)
IA  660 :
CM  670 rem goto the sprite subroutine,
KJ  680 rem turn number nu into a sprite
HM  690 rem and display it in motion
JL  700 gosub 1000
KD  710 :
LM  720 rem wait a while, then try another
OB  730 rem number
CK  740 ford = 1000to2000step5:pokesx,d/10
IN  745 nextd:pokesn,0:pokesx,100:goto610
CG  750 :
OD  760 rem = = = = = end of mainline = = = =
GH  770 :
AI  780 :
JL  900 rem----sprite number subroutine
CA  910 :
FC  920 rem---turn the sprite off
DN  1000 poke sn,0
GG  1010 :
DM  1020 rem---clear part of sprite
FB  1030 fori = 832to852:pokei,0:nexti
EI  1040 :
ML  1050 rem---make nu a string 3 characters
NJ  1060 rem in length with leading spaces
HK  1070 nu$ = right$("  " + str$(nu),3)
MK  1080 :
BA  1090 rem---select the 3 digit
NJ  1100 rem characters, one at a time
KF  1110 rem and change the character into
NF  1120 rem a number nd, skip spaces
JM  1130 for ch = 0to2:in = 0
GG  1140 ch$ = mid$(nu$,(ch + 1),1)
OI  1150 if ch$ = "  "then 1230
MC  1160 nd = val(ch$)
GE  1170 rem---get the character pattern for
DI  1180 rem      this number and place it in
FL  1190 rem      the proper sprite memory area
AP  1200 for i = (832 + ch)to(855 + ch)step3
IM  1210 poke i,peek(in + cm + nd*8):in = in + 1
DK  1220 next i
LL  1230 next ch
ME  1240 :
HA  1250 rem---turn finished sprite on and
OF  1260 rem return to main program
JM  1270 poke sn,1:return
```

Adding Depth To Your Screens

Stacy McInnis
Upland, California

- Some Tips on 3-D Sprite Animation

Artists use various techniques to make their flat canvasses appear to have depth. When you begin to draw a screen for your sprites to move upon you can use many of those same techniques. But you have an additional advantage in adding a three dimensional appearance to your screen. Your sprites can move and change size and colour.

For discussion, assume your game takes place in a small town which is being invaded by monsters. First let us look at the background screen. Buildings and trees should be larger near the bottom of the screen and smaller towards the top. This will help give the illusion that the bottom of the screen is nearer to the viewer. Openings in a building can be used to give depth as our monsters can be seen passing behind them. Images that overlap also give depth to a picture. For instance, a tree can be placed in front of a building with its base lower than the base of the building. This will give the illusion that the tree is closer than the building. Either designing your own character graphics set or building a bit map screen will enable you to more readily use such techniques as roads that narrow in the distance, that is as they move from the bottom of the screen to the top, and buildings in the distance that are smaller and contain smaller windows.

Once your background is set you are ready to move your sprites onto the screen. Remember our monsters attacking the village? Set sprite zero to our laser gun fire. It only shows when you push the joystick button. Sprite one is our fighter plane that has flown in to protect the village from the dreaded monsters. Since we have eight sprites available, this leaves us with six monster sprites to move on the screen.

If your six monsters are to move horizontally across the screen, make sure that monster sprite 2 is on the bottom of the screen. Sprite 3 is next and sprite 7 is the highest. When the sprites cross each other's paths, the sprite with the lowest sprite number remains visible, obscuring the higher-numbered sprites as if they were behind it.

Another method to add depth is to change the size of your monster as he moves down the screen. This is particularly effective if the sprite is to appear to be attacking from a distance and is moving very fast towards you. The sprite may be made to change its size as it passes a given position on the screen (as the X and Y co-ordinates in $D000 to $D011 change). In general, the sprite should become larger as it approaches the bottom of the screen. The easiest way to change the sprite size is by using the vertical expansion register (location $D017) and the horizontal expansion register (location $D01D). When the sprite first appears, the bits in the expansion register for the sprite should be zero to set the sprite to the regular size. Then as the sprite passes a selected point on the screen, you can set the expansion bits and have the sprite double in size. Depending on the speed and shape of the sprite, you may want to first set only the horizontal bit, then expand vertically further down the screen. Experiment to find the best effect.

Alternatively to working with the horizontal and vertical registers, you may change your sprite size by using different sprite definitions or images. To do this, change the appropriate sprite data pointer to point to a new definition. Each new definition should be a similar image but a different size. This of course involves more time in creating the sprites and more memory to store them than just using the expansion registers. Three different sizes for attacking monsters seem sufficient but more could certainly be used to create a smoother transition.

Another method to give an impression of depth is the use of colour. Assume that you have a black background. Your sprite could show up first as dark grey, then as it approaches, turn light grey, then lighter and finally white or yellow. It is even more effective if the sprite is multicolour and eventually appears a brilliant combination of yellow and red. You can use similar techniques in reverse to show a monster retreating into the distance. However, the nature of monsters is such that they tend to attack.

Use you sprite/background display priority (location $D01B) to control whether a sprite moves in front of or behind the background. If a sprite is supposed to be at a great distance then it should move behind the background. In our example, the background would be the trees and buildings. If a sprite is close it should move in front of the village. It is most effective if some sprites move in front while others move in the background. Since you know the location of your sprite, you can change its background display priority as a function of where it is on the screen. However, before you do this be sure and check the sprite/background collision register at $D01F or you may cause a monster to cut right through the middle of a building.

With the above ideas in mind you may add a new dimension to your graphics screens.

Viewports For The Commodore 64

Anthony Bryant
Winnipeg, Manitoba

A Hi-res and Multicolour Mode Windowing Utility

In Volume 5, Issue 06 of The Transactor, Gary Kiziak presented his 'HIRES' graphics utility package for the C–64. That package gives the programmer an array of flexible drawing commands. HIRES resides in the free RAM at $C000–$C81E and uses direct SYS calls to plot on the bitmap at $E000. Viewports works in conjunction with HIRES adding additional commands, in like manner, and resides at $C864–$CBFF.

If at any one time the programmer would like to devote his attention to a particular area of the hires screen, the new command, VIEW, allows the definition of subsets of the screen, called VIEWPORTS. Once a viewport has been defined, all subsequent drawing appears within that viewport. VIEW compresses the old screen coordinates to fit into the new viewport. Therefore, scaling is easily accomplished.

Refer to the article by Gary Kiziak for details on the screen coordinates, character cells and HIRES commands. Viewports closely follows that structure, style and labelling.

'VIEWPORTS.BAS' is a BASIC loader that, when run, will create a program file called 'VIEWPORTS' on your disk. (Make it the same disk that holds 'HIRES') This is the machine code that contains VIEW and a number of supporting commands which makes Viewports quite powerful.

Editor's Note: *Gary Kiziak's HIRES utility has found it's way into yet another application. The 'HIRES' create loader is not printed here, but the loadable 'HIRES' file will be included on Transactor Disk #13 for this issue. For a complete description of the Kiziak HIRES utility, see Volume 5, Issue 06, titled "Aids and Utilities".*

'DEMO 1' and 'DEMO 2' are sample Basic programs that demonstrate the new commands. Before looking at these, let's go through the new commands, their syntax and the various options available.

1. Setting Up A Hi–Res Viewport

To define a viewport, a line with the following syntax is required.

```
100 SYS VIEW,LC,TR,WIDTH,DEPTH
```

where:

LC	is the left–most column	(0–39)
TR	is the top–most row	(0–24)
WIDTH	is the number of columns wide	(1–40)
DEPTH	is the number of rows deep	(1–25)

A viewport is set up on character cell boundaries and is similar to the make up of a normal text screen but, once defined, it behaves like a hires screen.

LC,TR is the top left–hand corner of the viewport, which, if you were to define the whole screen as a viewport, would be 0,0. eg:

```
SYS VIEW,0,0,40,25
```

There is a shorter way:

```
SYS VIEW
```

. . .with no parameters defines the whole screen as a viewport.

HIRES uses the Cartesian system of coordinates where point x,y equal to (0,0) is the lower left hand corner and x,y equal to (319,199) <or (159,199) for multicolour mode> is the top right hand corner of the hires screen. Wherever a viewport is located, point (0,0) is the lower left hand corner of the viewport, and point (319,199) <or (159,199) for multicolour mode> is the top right hand corner of the viewport.

Now, plotting using HIRES commands PLOT, MOVE, DRAW, and BOX is scaled automatically to fit the viewport. All of the HIRES commands keep their original syntax, so old programs which used these commands can be viewported.

It is also possible to fill a viewport with new colours by adding on colour parameters with a line like:

```
100 SYS VIEW,LC,TR,WIDTH,DEPTH,BG,C1
    (in hires mode)
```

and

```
100 SYS VIEW,LC,TR,WIDTH,DEPTH,BG,C1,C2,C3
    (in multicolour mode)
```

where BG, C1, C2 and C3 are the backgound and plotting colours from HIRES.

2. Colouring The Viewport

When these extra colour parameters are included in a VIEW command, they are passed on to VCOLOUR, the next command.

```
200 SYS VCOLOUR,BG,C1
    (in hires mode)
```

and

```
200 SYS VCOLOUR,BG,C1,C2,C3
    (in multicolour mode)
```

VCOLOUR actually changes the background and plotting colours, filling the screen colour memory WITHIN the currently defined viewport.

3. This Is A Frame Up

With this simple command:

```
300 SYS VFRAME
```

A box is drawn around the perimeter of the currently defined viewport.

```
300 SYS VFRAME,C,M
```

also works, as the frame is drawn in the currently active plotting colour, where C is the SELPC and M is the DMODE. (See the HIRES BOX command)

4. A Wipe Out

New commands VIEW and VCOLOUR do not affect the bitmap, but there are occasions when you would want a clean viewport amidst a cluttered hires screen.

```
400 SYS VWIPE
```

clears the bitmap within the currently defined viewport.

HIRES commands draw on the bitmap located at $E000 (under the Kernal ROM). Viewports allows for an alternate bitmap located at $A000 (under the BASIC ROM).

```
400 SYS VWIPE,1
```

clears the alternate bitmap (unseen) – an area the same dimensions as the currently defined viewport. Note: ',1' is merely a flag and could be ',' anything between 0–255.

5. A Bitmap Swap

The alternate bitmap is useful for saving viewports or parts of viewports.

```
500 SYS VSWAP
```

does a byte for byte exchange of the HIRES bitmap with the Viewports alternate bitmap. Only that portion of the bitmaps within the currently defined viewport is swapped. Consequently, smaller viewports take less time to swap and wipe and multiple, overlapping viewports are possible.

6. Scroll Your Rows

```
600 SYS VUP
```

scrolls the currently defined viewport up one row throwing away the top row and filling the bottom row with blanks.

```
600 SYS VUP,1
```

changes the scroll to the wrap around type, where the top row is moved to the bottom row after all rows are moved up.

7. Scrolling Columns

```
700 SYS VLFT
```

scrolls the currently defined viewport left one column, throwing away the left most column and filling blanks into the right column.

```
700 SYS VLFT,1
```

changes the scroll to the wrap around type, where the left–most column is moved to the right–most column after all columns are moved left.

Only that portion of the bitmap within the viewport is scrolled, so narrower viewports scroll faster. In addition, screen colour memory is NOT scrolled, so it is desirable to keep to a uniform colour scheme within a viewport when scrolling.

Technical Notes

Viewports patches directly into the HIRES routines, modifying PLOT, MOVE, DRAW and BOX to allow scaling. Scaling is accomplished using l6 bit integer math routines (not BASIC's floating point routines) so it is very fast and does not slow down PLOT, MOVE, DRAW or BOX much. If SYS VIEW is used without parameters (or the equivalent SYS VIEW,0,0,40,25) so that the whole screen is defined as the viewport, then the scaling math routines are simply bypassed and the slow down is now only a few machine cycles.

The new commands make use of many routines from HIRES. For example, VFRAME sets up the x,y parameters and then jumps into the BOX command to do the actual plotting. So if HIRES isn't in place, the new commands will crash.

Viewports occupies free RAM space below HIRES, at $C864–$CBFF and, like HIRES, the first memory addresses comprise the command jump table. Besides the use of the alternate bitmap memory area RAM from $A000–$BFFF, a portion of RAM from $DE00–$DF40 is used in the scrolling routines.

'DEMO 1'

'DEMO 1' is a demonstration program that shows how easy it is to scale an existing program (in this case, Gary Kiziak's pie chart making program) and manipulate multiple viewports. It's just a matter of setting up a viewport and GOSUBing to the original pie drawing subroutines. The do–nothing FOR/NEXT loops just slow down the demo.

'DEMO 2'

'DEMO 2' does a simulation of a hires menu manager. Computers like the Macintosh don't have text screens and must communicate only using hires screens.

Experimenting with HIRES VIEWPORTS may lead you down new and exciting paths.

Basic Loader To Create Object File On Disk

```
LM 1000 rem save"0:viewports.bas",8
JD 1010 rem ** an enhancement package
GF 1020 rem ** for garry kiziak's 'hires'
CF 1030 rem ** by anthony  bryant
LB 1040 rem ** winnipeg, manitoba
OI 1050 :
MI 1060 rem ** this program will create
JO 1070 rem ** a load and run module on
EG 1080 rem ** disk called 'viewports'
GL 1090 :
FO 1100 open 15,8,15: open 8,8,1,"0:viewports"
LN 1110 input#15,e,e$,b,c: if e then close 15:
        print e;e$;b;c: stop
EN 1120 :
ME 1130 for j=51300 to 52219: read x: print#8,chr$(x);:
        ch=ch+x: next: close8
EH 1140 if ch<>119481 then print "checksum
        error!": stop
PD 1150 print "** module created **": end
MP 1160 :
CC 1170 data 100, 200,  76, 156, 201,  76,  17, 202
ON 1180 data  76,  88, 202,  76, 243, 202,  76, 246
PM 1190 data 202,  76,  59, 203,  76,  62, 203,   0
DI 1200 data   0,  40,  25,   0,   0,   0,   0,   0
JN 1210 data   0, 173, 125, 200, 174, 126, 200,  32
JK 1220 data  30, 201, 173, 122, 200,  10,  10,  10
CK 1230 data  73, 255,  24, 105, 200, 162,   0,  32
EF 1240 data  58, 201,  32, 113, 194,  76, 125, 194
DI 1250 data 169,   0, 141, 121, 200, 141, 122, 200
HC 1260 data 169,  40, 141, 123, 200, 169,  25, 141
JB 1270 data 124, 200,  96, 169, 218, 160, 200, 141
GE 1280 data 111, 194, 140, 112, 194, 141, 248, 195
OD 1290 data 140, 249, 195, 169, 221, 160, 200, 141
PC 1300 data  10, 196, 140,  11, 196, 169, 250, 160
DN 1310 data 200, 141,  71, 197, 140,  72, 197,  96
PD 1320 data  32, 253, 174,  32, 138, 193,  32,   9
CM 1330 data 201,  24, 109, 125, 200, 144,   1, 232
HA 1340 data  32,  30, 201,  32,  37, 201,  24, 109
LK 1350 data 127, 200,  32,  58, 201,  76, 156, 193
HG 1360 data  32, 253, 174,  32, 138, 193,  32,   9
EL 1370 data 201,  32,  37, 201,  76, 156, 193, 173
EI 1380 data  43, 192, 174,  44, 192, 172, 123, 200
NH 1390 data 192,  40, 240,   8,  32,  65, 201, 160
MJ 1400 data  40,  32, 103, 201, 141,  43, 192, 142
BM 1410 data  44, 192,  96, 173,  45, 192, 174,  46
EE 1420 data 192, 172, 124, 200, 192,  25, 240,   8
BH 1430 data  32,  65, 201, 160,  25,  32, 103, 201
KE 1440 data 141,  45, 192, 142,  46, 192,  96, 133
JK 1450 data  20, 134,  21, 132,  34, 132,  36, 169
GI 1460 data   0, 162,   8,  70,  34, 144,   3,  24
LK 1470 data 101,  20, 106, 102,  34, 202, 208, 245
HL 1480 data 166,  21, 240,   3,  24, 101,  36, 133
HA 1490 data  35, 170, 165,  34,  96, 133,  20, 134
HN 1500 data  21, 132,  34, 169,   0, 133,  36, 133
PJ 1510 data  37, 162,  16,  38,  20,  38,  21,  38
JH 1520 data  36,  38,  37,  56, 165,  36, 229,  34
AB 1530 data 168, 165,  37, 233,   0, 144,   4, 132
HC 1540 data  36, 133,  37, 202, 208, 229,  38,  20
OF 1550 data  38,  21, 166,  21, 165,  20,  96,  76
AI 1560 data  72, 178,  32, 162, 200,  32, 181, 200
EB 1570 data  32, 121,   0, 240,  54,  32, 241, 183
CG 1580 data 224,  40, 176, 235, 142, 121, 200,  32
AG 1590 data 241, 183, 224,  25, 176, 225, 142, 122
BJ 1600 data 200,  32, 241, 183, 138, 240, 216,  24
CD 1610 data 109, 121, 200, 201,  41, 176, 208, 142
BI 1620 data 123, 200,  32, 241, 183, 138, 240, 199
FI 1630 data  24, 109, 122, 200, 201,  26, 176, 191
NE 1640 data 142, 124, 200, 173, 121, 200,  10,  10
HI 1650 data 141, 125, 200, 162,   0,  44,  53, 192
NL 1660 data  48,  10, 173, 125, 200,  10, 141, 125
KE 1670 data 200, 144,   1, 232, 142, 126, 200, 173
NK 1680 data 124, 200,  24, 109, 122, 200,  73, 255
MH 1690 data  24, 105,  26,  10,  10,  10, 141, 127
IB 1700 data 200,  32, 121,   0, 208,   1,  96,  32
GC 1710 data 241, 183, 138,  41,  15, 141,  61, 192
DI 1720 data  32,  11, 197,  32, 131, 200,  44,  53
HN 1730 data 192,  16,  15, 173,  61, 192, 141,  33
DF 1740 data 208, 173,  62, 192,  13,  63, 192, 141
MC 1750 data  70, 192, 162,   0, 142,  66, 192, 160
KN 1760 data   0,  32,  75, 195, 200, 204, 123, 200
PC 1770 data 144, 247,  24, 165, 253, 105,  40, 133
NN 1780 data 253, 144,   2, 230, 254, 232, 236, 124
BI 1790 data 200, 144, 228,  76,  51, 197,  32, 131
LE 1800 data 200, 173, 124, 200,  10,  10,  10, 168
LC 1810 data 136, 152, 162,   0,  32,  58, 201, 173
FL 1820 data 123, 200,  10,  10,  10, 144,   1, 232
LB 1830 data  56, 233,   1, 168, 138, 233,   0, 170
OF 1840 data 152,  32,  30, 201,  76,  73, 197, 160
JB 1850 data   0, 173, 128, 200, 240,  15,  48,   3
EK 1860 data 177, 253,  72, 177, 251, 145, 253, 173
AL 1870 data 128, 200,  48,   3, 104, 145, 251, 200
KM 1880 data 192,   8, 208, 229, 152,  24, 101, 251
PK 1890 data 133, 251, 144,   2, 230, 252, 152,  24
HG 1900 data 101, 253, 133, 253, 144,   2, 230, 254
MF 1910 data 232, 236, 123, 200, 144, 201,  96, 132
PH 1920 data   2, 162,   0, 165, 251, 133,  91, 165
GP 1930 data 252, 133,  92, 165, 253, 133,  93, 165
HB 1940 data 254, 133,  94,  32, 129, 202,  24, 165
DB 1950 data  91, 105,  64, 133, 251, 165,  92, 105
KH 1960 data   1, 133, 252,  24, 165,  93, 105,  64
EF 1970 data 133, 253, 165,  94, 105,   1, 133, 254
FJ 1980 data 164,   2, 200, 204, 124, 200, 144, 199
HC 1990 data  96, 169,   0,  44, 169,   1, 141, 128
OK 2000 data 200,  32,  30, 203, 166, 251, 165, 252
AC 2010 data  41, 191, 134, 253, 133, 254, 172, 128
OL 2020 data 200, 208,   9, 172, 129, 200, 240,   4
EG 2030 data 134, 251, 133, 252, 160,   0,  32, 185
NC 2040 data 202,  76, 218, 192,  32, 121,   0, 240
JB 2050 data   6,  32, 241, 183, 169,   1,  44, 169
CM 2060 data   0, 141, 129, 200,  32, 131, 200,  32
```

```
GO 2070 data 201, 192, 165,   1,  41, 248, 133,   1
LG 2080 data  96, 169,   0,  44, 169,   1, 141, 130
JD 2090 data 200, 169, 255, 141, 128, 200,  32,  30
OL 2100 data 203, 173, 130, 200, 208,  66, 162,   0
MH 2110 data 169, 222, 134, 253, 133, 254, 166, 251
LK 2120 data 165, 252, 134,  97, 133,  98, 172, 124
JB 2130 data 200,  32, 185, 202, 166,  97, 165,  98
JN 2140 data 134, 253, 133, 254, 160,   1,  32, 185
IJ 2150 data 202, 166, 253, 165, 254, 134, 251, 133
LG 2160 data 252, 162,   0, 169, 222, 134, 253, 133
AJ 2170 data 254, 173, 129, 200, 141, 128, 200, 172
DL 2180 data 124, 200,  32, 185, 202,  76, 218, 192
CH 2190 data 160,   0, 132,   2, 165, 251, 133,  91
ND 2200 data 165, 252, 133,  92, 162,   0, 169, 222
IO 2210 data 134, 253, 133, 254, 166, 251, 165, 252
PD 2220 data 134,  97, 133,  98, 174, 123, 200,  32
KB 2230 data 129, 202, 166,  97, 165,  98, 134, 253
LC 2240 data 133, 254, 162,   1,  32, 129, 202, 166
MA 2250 data 253, 165, 254, 134, 251, 133, 252, 162
GL 2260 data   0, 169, 222, 134, 253, 133, 254, 173
KM 2270 data 129, 200, 141, 128, 200, 174, 123, 200
IC 2280 data  32, 129, 202, 169, 255, 141, 128, 200
NG 2290 data  24, 165,  91, 105,  64, 133, 251, 165
NM 2300 data  92, 105,   1, 133, 252, 164,   2, 200
JB 2310 data 204, 124, 200, 144, 157,  76, 218, 192
```

Demonstration Program Number 1

```
JE 100 rem save " 0:demo 1 " ,8
BH 110 rem 'viewports' demonstration
KH 120 if peek(49153)<>194 then load " 0:hires " ,8,1
GM 130 if peek(51301)<>156 then load " 0:viewports " ,8,1
AA 140 :
KE 150 lc$ = chr$(14): cd$ = chr$(17): rv$ = chr$(18)
EB 160 :
PO 170 rem 'hires' variables
LA 180 hires = 49152: draw = hi + 3: plot = dr + 3
OM 190 move = pl + 3: clscr = mo + 3: dmode = cl + 3
MF 200 selpc = dm + 3: colour = se + 3: box = co + 3
BC 210 text = bo + 3: prnt = te + 3: chset = pr + 3
LC 220 trap = ch + 3
KF 230 :
AI 240 rem 'viewport' variables
KC 250 view = 51300: vcol = vi + 3: vframe = vc + 3
MA 260 vwipe = vf + 3: vswap = vw + 3: vup = vs + 3
EN 270 vlft = vu + 3
MI 280 :
MN 290 rem begin the demonstration
AA 300 c = 0: r = 0: width = 40: depth = 25: bg = 1: c1 = 6
NB 310 a$ = a$ + " "
GN 320 sys hires,0,bg,c1: poke 53280,1
EB 330 sys view: sys vwipe,1
PH 340 a$ = lc$ + " Pie Charts are Easy "
JJ 350 a$ = cd$ + rv$ + " [10 spcs] " + a$ +
       rv$ + " [11 spcs] "
OJ 360 gosub630 'frame & do big pie chart
JC 370 for i = 1to1000: next
NI 380 width = 19: depth = 12
LD 390 c = 0: r = 0: bg = 7: c1 = 0
MN 400 a$ = lc$ + rv$ + " [4 spcs]Viewport #1[4 spcs] "
GM 410 gosub610 'frame & scale pie chart
PC 420 depth = 16
CP 430 c = 21: r = 0: bg = 3: c1 = 0
HA 440 a$ = lc$ + rv$ + " [4 spcs]Viewport #2[4 spcs] "
OO 450 gosub610 'frame & scale pie chart
DI 460 width = 34: depth = 8
MH 470 c = 3: r = 17: bg = 4: c1 = 0
CD 480 a$ = lc$ + rv$ + " [4 spcs]Viewport #3[4 spcs] "
GB 490 gosub610 'frame & scale pie chart
LK 500 for i = 1to1000: next
CH 510 :
GA 520 rem now swap viewports back
DI 530 bg = 1: c1 = 6
KM 540 sys view, 0,0,19,12,bg,c1: sys vswap
IJ 550 sys view,21,0,19,16,bg,c1: sys vswap
MA 560 sys view,3,17,34, 8,bg,c1: sys vswap
AC 570 wait 198,1: get a$: sys text
EE 580 end
CM 590 :
JH 600 rem the pie drawing subroutines
DE 610 sys view,c,r,width,depth,bg,c1
PL 620 sys vswap
LE 630 sys vframe: sys prnt,c,r,a$
HD 640 sys colour,c1
KF 650 xc = 159: yc = 100: xr = 70: yr = 50: inc = 10
LE 660 sa = 45: ea = 75: gosub870
EA 670 sa = 75: ea = 160: gosub870
PD 680 sa = 160: ea = 240: gosub870
CF 690 sa = 240: ea = 325: gosub870
GD 700 xc = 175: sa = -35: ea = 45: gosub870
JP 710 sys box,14,170,291,160
IC 720 sys box,8,174,305,168
GP 730 return
IF 740 :
NF 750 rem draw arc
GI 760 z1 = sa*π/180: z2 = ea*π/180: z3 = inc*π/180
BJ 770 x = xc + xr*cos(z1): y = yc + yr*sin(z1)
NA 780 sys move,x,y
JC 790 for i = z1 to z2 step z3
EM 800 x = xc + xr*cos(i): y = yc + yr*sin(i)
LC 810 sys draw,x,y
ID 820 next
ED 830 sys draw,xc + xr*cos(z2),yc + yr*sin(z2)
EG 840 return
GM 850 :
GN 860 rem draw pie
NC 870 gosub760
FE 880 sys draw,xc,yc
OG 890 sys draw,xc + xr*cos(z1),yc + yr*sin(z1)
AK 900 return
```

Demonstration Program Number 2

```
LE  100 rem save "0:demo 2",8
BH  110 rem 'viewports' demonstration
CN  120 if peek(49153)<>194 then load "hires",8,1
CC  130 if peek(51301)<>156 then load "viewports",8,1
AA  140 :
PG  150 lc$=chr$(14): ro$=chr$(146)
EB  160 :
PO  170 rem 'hires' variables
LA  180 hires=49152: draw=hi+3: plot=dr+3
OM  190 move=pl+3: clscr=mo+3: dmode=cl+3
MF  200 selpc=dm+3: colour=se+3: box=co+3
BC  210 text=bo+3: prnt=te+3: chset=pr+3
LC  220 trap=ch+3
KF  230 :
AI  240 rem 'viewport' variables
KC  250 view=51300: vcol=vi+3: vframe=vc+3
MA  260 vwipe=vf+3: vswap=vw+3: vup=vs+3
EN  270 vlft=vu+3
MI  280 :
AH  290 rem a hires menu manager simulation
BJ  300 bg=1: c1=0
EN  310 sys hires,0,bg,c1: poke 53280,3
KA  320 sys view: sys vwipe,1
OL  330 :
HB  340 rem set up menu command line
NA  350 c=0: r=0: sys view,c,r,40,1,bg,c1
OO  360 a$=lc$+ro$+" Draw[3 spcs]File
        [3 spcs]View[3 spcs]Options"
KI  370 c=3: sys prnt,c,r,a$
AP  380 :
PL  390 rem set up green workbench
BF  400 c=0: r=1: sys view,c,r,40,24,13,c1
PL  410 sys vframe
IB  420 :
PB  430 rem select menu item 'draw'
LH  440 c=2: r=0: sys view,c,r,6,1,0,1
KJ  450 r=1: sys view,c,r,9,8,14,1
FB  460 sys vswap: sys vframe
FH  470 c=3: r=3: sys prnt,c,r,lc$+"Circles"
GE  480 r=4: sys prnt,c,r,lc$+"Boxes"
NN  490 r=5: sys prnt,c,r,lc$+"Arcs"
CE  500 r=6: sys prnt,c,r,lc$+"Lines"
FL  510 for i=1to1000: next
LI  520 sys view,c,r,7,1,1,6
JM  530 for i=1to1000: next
LI  540 c=2: r=1: sys view,c,r,9,8,13,c1
GP  550 sys vswap: sys vwipe,1
JN  560 c=2: r=0: sys view,c,r,6,1,bg,c1
OK  570 :
CG  580 rem return to green workbench
PA  590 c=0: r=1: sys view,c,r,40,24,13,c1
HG  600 sys move,0,0
CO  610 for j=1to50
JE  620 sys draw,rnd(1)*320,rnd(1)*200
IF  630 next j
HD  640 for i=1to1000: next
OP  650 :
JP  660 rem select menu item "file"
NH  670 c=9: r=0: sys view,c,r,6,1,0,1
CJ  680 r=1: sys view,c,r,20,20,3,c1
LP  690 sys vswap: sys vframe
AE  700 c=10: r=2: sys view,c,r,18,18
HN  710 r=18: gosub820 'print filenames
HI  720 for i=1to1000: next
MG  730 c=9: r=15: sys view,c,r,20,1,0,3
LJ  740 for i=1to1000: next
LL  750 c=9: r=1: sys view,c,r,20,20,13,c1
LE  760 sys vswap
DO  770 c=9: r=0: sys view,c,r,6,1,1,0
CP  780 wait198,1: get a$: sys text
GB  790 end
EJ  800 :
GH  810 rem make up some filenames & scroll
HC  820 a$="> dir      [13 spcs]      ": gosub970
DH  830 a$="           [19 spcs]      ": gosub970
JE  840 a$="C-64 Wedge                ": gosub970
CK  850 a$="  Dos 5.1                 ": gosub970
OB  860 a$="Display T & S              ": gosub970
LH  870 a$="Unicopy64                 ": gosub970
BD  880 a$="Supermon64.V1             ": gosub970
OM  890 a$="-------------------       ": gosub970
IA  900 a$="Commands.text demo        ": gosub970
EF  910 a$="  Commands.ml             ": gosub970
CJ  920 a$="Hires demo program        ": gosub970
PH  930 a$="  Hires.ml                ": gosub970
MO  940 a$="Viewports demo prg        ": gosub970
GL  950 a$="  Viewports.ml            ": gosub970
CP  960 a$="  . . . . . .             "
LN  970 sys prnt,c,r,a$: sys vup
PG  980 for i=1to100: next
KP  990 return
```

Commodore 64 High Resolution Draw Routine

David Jankowski
Cairns, Australia

A High-Res Utility You Can Incorporate Into Your Own Programs

This interesting ML Hi–Res routine resides at $C000 (49152) and is invoked by passing a string variable from your basic program with SYS 49152,A$.

The Hi–Res draw routine can be used with keyboard input, joystick or drawing pad.

Listing 1 creates the ML program which actually plots your drawing on the Hi–Res screen.

Listing 2 and 3 give you some practical examples how you could use Hi–Res Draw from within your own Basic programs and Listing 4 is a complete joystick drawing program ready for use.

Commands

B Moves Pixel Cursor X pixel(s) DOWN and LEFT. B+chr$(X) where X = 1–199
C CLEARS the High–Res screen.
D Moves Pixel Cursor X pixel(s) DOWN. D+chr$(X) where X = 1–199
H Moves Pixel Cursor X pixel(s) UP and LEFT. H+chr$(X) where X = 1–199
J Moves Pixel Cursor X pixel(s) UP and RIGHT. J+chr$(X) where X = 1–199
K Sets pixel KOLOUR to X and screen KOLOUR to Y. K+chr$(X)+chr$(Y) where X and Y = 0–15
L Moves Pixel Cursor X pixel(s) LEFT. L+chr$(X) where X = 1–255 (repeat to maximum 320 pixel positions).
N Moves Pixel Cursor X pixel(s) DOWN and RIGHT. N+chr$(X) where X = 1–199
O Starts a repeating LOOP. (i.e. O+chr$(100)+L+chr$(10)+D+chr$(1)+Z. This draws a BAR, 10 pixels wide, 100 pixels down from the current cursor position.)
P PLOT. Plots a pixel anywhere on the screen. P+chr$(X1)+chr$(X2)+chr$(Y) where X1 = 1–255, X2 = 0–1 and Y = 1–199. Note: X1 should not be greater than 64 when X2 = 1. (i.e. 64+(1x255)=X1+(X2)=320)
R Moves Pixel Cursor X pixel(s) RIGHT. R+chr$(X) where X = 1–255 (repeat to maximum 320 pixel positions)
U Moves Pixel Cursor X pixel(s) UP. U+chr$(X) where X = 1–199
Z Indicates 'NEXT' in loop started with O.
0 Pen ERASE. Pen draws in screen colour.
1 Pen DOWN. Pen draws in cursor colour.
2 Pen UP. Pen does NOT draw. Cursor may be moved to anywhere on the screen.
+ Cursor OFF. Switches Cursor OFF.
– Cursor ON. Switches Cursor ON.

Listing 1: Hi–Res 64 Basic Loader

```
EG 1000 rem save" 0:hi-res 64.bas" ,8
LM 1010 rem ** written by: david jankowski, cairns
           qld 4870, australia
AH 1020 :
AK 1030 for j=49152 to 49920: read x: poke j,x:
           ch=ch+x: next
CB 1040 if ch<>89345 then print " checksum error " :
           stop
AA 1050 print " data okay – read article to use" : end
IJ 1060 :
BM 1070 data  32, 253, 174,  32, 158, 173,  32, 163
EP 1080 data 182, 134, 253, 132, 254,  56, 233,   1
PI 1090 data 141,  31, 195, 160, 255, 140,  30, 195
FC 1100 data  32,  58, 192, 201,  80, 208,  45,  32
JL 1110 data  58, 192, 141,  52,   3,  32,  58, 192
JM 1120 data 141,  53,   3,  32,  58, 192, 141,  54
HK 1130 data   3,  32,  61, 194,  76,  24, 192, 104
DA 1140 data 104,  96, 172,  30, 195, 173,  31, 195
JJ 1150 data 205,  30, 195, 240, 242, 200, 140,  30
JP 1160 data 195, 177, 253,  96, 201,  67, 208,   6
HL 1170 data  32, 200, 194,  76,  24, 192, 201,  48
FO 1180 data 208,   7, 169,   1, 133, 252,  76,  24
NM 1190 data 192, 201,  49, 208,   7, 169,   0, 133
CK 1200 data 252,  76,  24, 192, 201,  50, 208,   7
HM 1210 data 169,   2, 133, 252,  76,  24, 192, 201
JD 1220 data  75, 208,  37, 169,   0, 141, 255, 195
FL 1230 data  32,  58, 192,  13, 255, 195,  24,  42
NO 1240 data  24,  42,  24,  42,  24,  42, 141, 255
PG 1250 data 195,  32,  58, 192,  13, 255, 195, 141
IL 1260 data 255, 195,  32, 224, 194,  76,  24, 192
NG 1270 data 201,  68, 208,   6,  32,  30, 194,  76
MP 1280 data  24, 192, 201,  82, 208,   6,  32, 241
EB 1290 data 193,  76,  24, 192, 201,  76, 208,   6
DH 1300 data  32, 196, 193,  76,  24, 192, 201,  85
HB 1310 data 208,   6,  32, 165, 193,  76,  24, 192
BI 1320 data 201,  78, 208,  33,  32,  58, 192, 141
CJ 1330 data  33, 195, 162,   0, 142,  34, 195, 169
IL 1340 data   1,  32,  33, 194, 169,   1,  32, 244
OM 1350 data 193, 174,  34, 195, 232, 236,  33, 195
PM 1360 data 208, 234,  76,  24, 192, 201,  66, 208
BI 1370 data  33,  32,  58, 192, 141,  33, 195, 162
IL 1380 data   0, 142,  34, 195, 169,   1,  32,  33
CJ 1390 data 194, 169,   1,  32, 199, 193, 174,  34
FA 1400 data 195, 232, 236,  33, 195, 208, 234,  76
FJ 1410 data  24, 192, 201,  74, 208,  33,  32,  58
JN 1420 data 192, 141,  33, 195, 162,   0, 142,  34
DI 1430 data 195, 169,   1,  32, 168, 193, 169,   1
DP 1440 data  32, 244, 193, 174,  34, 195, 232, 236
EG 1450 data  33, 195, 208, 234,  76,  24, 192, 201
DN 1460 data  72, 208,  33,  32,  58, 192, 141,  33
KJ 1470 data 195, 162,   0, 142,  34, 195, 169,   1
CH 1480 data  32, 168, 193, 169,   1,  32, 199, 193
CE 1490 data 174,  34, 195, 232, 236,  33, 195, 208
JP 1500 data 234,  76,  24, 192, 201,  79, 208,  15
LO 1510 data  32,  58, 192, 141,  35, 195, 172,  30
MK 1520 data 195, 140,  36, 195,  76,  24, 192, 201
EH 1530 data  90, 208,  23, 172,  35, 195, 136, 140
```

```
DN 1540 data  35, 195, 192,   0, 208,   3,  76,  24
NN 1550 data 192, 172,  36, 195, 140,  30, 195,  76
KE 1560 data  24, 192, 201,  43, 208,   8, 169, 208
IM 1570 data 141, 181, 194,  76,  24, 192, 201,  45
EJ 1580 data 208,   8, 169, 240, 141, 181, 194,  76
DK 1590 data  24, 192,  76,   8, 175,  32,  58, 192
IJ 1600 data 141,  37, 195, 162,   0, 172,  54,   3
OE 1610 data 136, 140,  54,   3, 142,  38, 195,  32
BB 1620 data  61, 194, 174,  38, 195, 232, 236,  37
PP 1630 data 195, 208, 234,  96,  32,  58, 192, 141
FC 1640 data  37, 195, 162,   0, 173,  52,   3,  56
IG 1650 data 233,   1, 144,  19, 141,  52,   3, 142
EN 1660 data  38, 195,  32,  61, 194, 174,  38, 195
IP 1670 data 232, 236,  37, 195, 208, 230,  96, 172
GA 1680 data  53,   3, 136, 140,  53,   3,  76, 212
EC 1690 data 193,  32,  58, 192, 141,  37, 195, 162
FK 1700 data   0, 173,  52,   3,  24, 105,   1, 176
LB 1710 data  19, 141,  52,   3, 142,  38, 195,  32
FH 1720 data  61, 194, 174,  38, 195, 232, 236,  37
KA 1730 data 195, 208, 230,  96, 172,  53,   3, 200
BA 1740 data 140,  53,   3,  76,   1, 194,  32,  58
OD 1750 data 192, 141,  37, 195, 162,   0, 172,  54
FO 1760 data   3, 200, 140,  54,   3, 142,  38, 195
CG 1770 data  32,  61, 194, 174,  38, 195, 232, 236
IG 1780 data  37, 195, 208, 234,  96, 166, 252, 224
KE 1790 data   2, 208,   1,  96, 173,  53,   3, 172
OC 1800 data  52,   3, 133,   5, 152,  41, 248, 133
ID 1810 data   4, 152,  41,   7, 133,   6, 173,  55
LE 1820 data   3, 172,  54,   3, 152,  74,  74,  74
HA 1830 data 133,   3, 152,  41, 248, 133,   2, 169
IP 1840 data   0, 133, 251, 162,   3,   6,   2,  38
FK 1850 data 251, 202, 208, 249, 165,   3,  24, 101
HE 1860 data 251, 133,   3, 152,  41,   7,  24, 101
CA 1870 data   2, 133,   2, 144,   2, 230,   3,  24
IK 1880 data 101,   4, 133,   2, 165,   3, 105,  32
AF 1890 data 133,   3, 165,   3, 101,   5, 133,   3
AH 1900 data 166,   6, 232, 169,   0,  56, 106, 202
PB 1910 data 208, 252, 166, 252, 240,  27, 160,   0
EK 1920 data  72,  81,   2, 145,   2, 169,   0, 133
KH 1930 data 162, 165, 162, 201,   2, 240, 250, 104
LD 1940 data  73, 255,  49,   2, 145,   2,  24, 144
MM 1950 data   6, 160,   0,  17,   2, 145,   2,  96
GD 1960 data 169,   0, 133,   2, 169,  32, 133,   3
MJ 1970 data 162,  32, 160,   0, 152, 145,   2, 136
BC 1980 data 208, 251, 230,   3, 202, 208, 246,  96
NM 1990 data 169,   0, 133,   2, 169,   4, 133,   3
EN 2000 data 169,   0, 141,   0,   4, 172, 255, 195
GO 2010 data 152, 162,   4, 160,   0, 145,   2, 136
JE 2020 data 208, 251, 230,   3, 202, 208, 246,  96
OI 2030 data 238
```

Listing 2: Bar-Graph Generator For Use With Hi-Res 64

```
NC 100 rem save " 0:hi-res 64 bargen " ,8
LG 110 rem ** bar graph generator **
LE 120 draw = 49152: hn = 0: print chr$(147)
PD 130 s$ = " -1ck " + chr$(0) + chr$(1) + " p "
       + chr$(0) + chr$(0) + chr$(199)
CE 140 print spc(14) " *bar graph* " : print
OM 150 print spc(14) " end with -1 "
GH 160 dim a(100): i = 0
HJ 170 input " bar value " ;a(i): if a(i)>hn then hn = a(i)
ML 180 if a(i)<>-1 then i = i + 1: goto 170
CD 190 :
FB 200 rem ** find height & width of bars **
LH 210 fa = 200/hn: x1 = int(320/i)
AF 220 :
JO 230 rem ** set up screen **
CP 240 poke 53265,peek(53265)or32:
       poke 53272,peek(53272)or8
NC 250 sys draw,s$
MN 260 for j = 0 to i-1
OD 270 fc = a(j)*fa: if fc<1 then fc = 1
GP 280 b$ = " o " + chr$(fc) + " r " + chr$(x1-2) + " l "
       + chr$(x1-2) + " u " + chr$(1) + " z "
LL 290 sys draw,b$: sys draw, " 2d " + chr$(fc) + " r "
       + chr$(x1) + " 1 " : next
HA 300 wait 198,1: poke 198,0: poke 53265,27:
       poke 53272,21: print chr$(147);
```

Listing 3: Letter Generator For Use With Hi-Res 64

```
OG 100 rem save " 0:hi-res 64 letter " ,8
NH 110 rem ** draw hi-res letters **
DH 120 input " letter size " ;lz: draw = 49152
GP 130 :
BF 140 rem ** set up hi res screen **
NN 150 s$ = " + ck " + chr$(3) + chr$(0) + " p " + chr$(10)
       + chr$(0) + chr$(0)
DN 160 sys draw,s$
MK 170 poke 53265,peek(53265)or32:
       poke 53272,peek(53272)or8
IC 180 :
DA 190 rem ** letter definitions **
KI 200 l1 = lz/2
HD 210 a$(1) = " 2r " + chr$(l1) + " 1b " + chr$(l1) + " d "
       + chr$(l1) + " u " + chr$(l1)
HE 220 a$(1) = a$(1) + " r " + chr$(l1*2) + " d "
       + chr$(l1) + " u " + chr$(l1) + " h " + chr$(l1)
MB 230 a$(1) = a$(1) + " 2r " + chr$(l1)
CG 240 sys draw,a$(1)
FN 250 wait 198,1: poke 198,0: poke 53265,27:
       poke 53272,21: print chr$(147);
```

Listing 4: Joystick Hi-Res Draw Routine

```
EK 100 rem save " 0:hi-res 64 joystk " ,8
JM 110 rem ** joystick hi-res draw rtn **
MO 120 :
MM 130 rem ** set up variables **
OI 140 poke 53265,peek(53265)or32:
       poke 53272,peek(53272)or8
KK 150 draw = 49152: a$(0) = " " : a$(1) = " u " + chr$(1):
       a$(2) = " d " + chr$(1): a$(3) = " "
CB 160 a$(4) = " l " + chr$(1): a$(5) = " h " + chr$(1):
       a$(6) = " b " + chr$(1): a$(7) = " "
NH 170 a$(8) = " r " + chr$(1): a$(9) = " j " + chr$(1):
       a$(10) = " n " + chr$(1)
DK 180 s$ = " + 1ck " + chr$(1) + chr$(0) + " p "
       + chr$(160) + chr$(0) + chr$(100)
BP 190 sys draw,s$
MD 200 :
MG 210 rem ** read the joystick **
IK 220 jv = peek(56320): fr = jv and 16:
       jv = 15-(jv and 15)
LC 230 if len(a$(jv)) then sys draw,a$(jv): goto 220
PB 240 if fr<>16 then s = 1-s: c$ = right$(str$(s),1):
       sys draw,c$
KJ 250 goto 220
```

Hi-Res Search and Print

Jack R. Farrah
Cincinnati, Ohio

The Complete Hires Printer Utility!

The Commodore 64 is capable of producing striking graphic images in high resolution mode and there are many good commercial and public domain programs available which support their creation. When you want to print these images, you're generally forced to do so from within the environment of the graphics package they were created with. If you have several of these, you're faced with a lot of wasted time booting up each separate program just to print the pictures.

For those of you with a Gemini SG10 printer and MW350 (or similar) interface, Hires Search and Print will allow you to print any bitmapped you've created and saved on disk. You can do this in either of two sizes, in positive or negative and as a mirror image. This gives you eight possible picture variations! Just type in and save the basic loader program and follow the instructions below.

After saving, load and run Hires Search and Print. After the Ready prompt, type NEW and then load any hires picture file that you wish to print. You should load it in as a relocatable file with:

LOAD "Filename",8 (don't use ,8,1)

Although the normal location of this file would probably require a relocating load, the ability of Hires Search and Print to deal with a wide variety of program files lies in being able to access all bitmap images in the same general area of memory. Therefore, all picture files are loaded into the basic program area. Once there, the bitmap image portion of the file can be located through Hires Search and Print's unique search function.

Some drawing packages may save pictures in multiple files. You are only interested in the file containing the bitmap image. Color information is not necessary for this program as we're dealing strictly with black and white. If you're not sure which file contains the bitmap, load either and you'll easily find if its the correct one.

After your picture file is loaded, hit the F5 key. Your screen now is in hires with a black and white image. Don't be alarmed if its a jumbled pattern of dots. If it is, its time to begin searching for the hires image. Use the cursor up/down key to cycle through memory. This key re-draws the hires screen starting one screen line further into memory each time its hit. Holding the key down will do this very quickly or just tap the key for slow cycling. You can go backwards through memory by using the SHIFT key with the up/down cursor. You can scan through memory again and again if you like, but the picture we're searching for will be within one or two screens of where we started. The reason you must search for the picture is because different drawing programs save not only the bitmap, but also color memory information in their files and the format of these files varies considerably. Some save the bitmap first, some last and some in between color data. As you're cycling, you will begin to see the hires picture appear. It will probably look out of registration and not be centered properly on the screen. Get as much of this image on the screen as you can with the cursor up/down key.

After "ballparking" the hires image, you're now ready to fine tune into proper resolution and screen centering. For this you use the left/right cursor key. Operation is identical to the up/down cursor except that now the screen is re-drawn starting just one byte further into memory each time. Again, using the SHIFT/cursor combination allows reverse movement in the event you overshoot the mark.

At any point in this process you can return to the text screen area of memory. Simply press the space bar to do so. If you have done a considerable amount of cursoring to find your picture, you may return to a blank screen and or one with no visible cursor. If the cursor is lost, just hit RUN STOP/RESTORE. This will disable Hires Search and Print so you'll need to:

SYS 49152

to reactivate it. None of this will affect the hires image you've previously located and you may return to it by pressing F5 again.

Once you've located a hires picture and oriented it on the display, you can load and display additional pictures created with the same drawing package and they too will be properly centered without the need for further searches, unless you have altered the setting while in hires mode.

Now let's print our picture. Turn on your Gemini printer and choose the image option you want from the list below.

a) F1 – prints a small, positive image of the hires screen (3.25" by 4.5")

b) F3 – prints a twice size, positive image (6.5" by 9")

c) Control and F1 or Control and F3 – prints a small or twice size negative image.

d) Shift and F1 or Shift and F3 – prints a small or twice size positive mirror image.

e) Shift,Control and F1 or F3 – prints a small or twice size negative, mirror image.

When using the negative and mirror options, be sure and have the Control and or Shift keys depressed before hitting the function key. To discontinue printing in the middle of a screen dump without shutting off the printer, just hit the Run/Stop key.

That's all there is to it! A printer utility that can handle all your hires needs in one compact package.

One final note. The program assumes that you are using a parallel printer interface with your Gemini printer that is put in transparent mode with a secondary address of 5. If your interface requires a different number, simply replace the Data number five at 49317 (Line 1260 of the BASIC Loader) with the correct number.

Hires Search and Print: BASIC Loader

```
LA 1000 rem hires search and print
AF 1010 rem jack r. farrah
FN 1020 for j= 49152 to 50032 : read x
BI 1030 poke j,x : ch = ch + x : next
HM 1040 if ch<> 123062  then print " checksum error "
        : end
ID 1050 rem
NF 1060 data 120, 169,  24, 141,  20,   3, 169, 192
IC 1070 data 141,  21,   3, 169,   0, 141, 233, 193
LI 1080 data 141, 253, 193, 141,   3, 194,  88,  96
KH 1090 data  76,  39, 192,  76, 228, 194,  76,  19
LO 1100 data 195,  76, 197, 194,  76, 106, 194, 165
BH 1110 data 197, 201,  64, 208,   8, 169,   0, 141
NK 1120 data 233, 193,  76,  49, 234, 174, 233, 193
KJ 1130 data 208, 248, 201,   6, 240, 230, 201,  60
JD 1140 data 240, 223, 201,   2, 240, 213, 201,   7
LC 1150 data 240, 212, 201,   4, 240,  24, 201,   5
CJ 1160 data 208, 224, 169,   1, 141, 253, 193, 169
GI 1170 data   0, 141, 254, 193, 169, 144, 141, 249
AL 1180 data 193, 169,   1, 141, 250, 193, 169,   1
NL 1190 data 141, 233, 193, 169,   0, 141, 252, 193
FB 1200 data 141,   0, 194, 173, 141,   2,  41,   5
GO 1210 data 240,  38, 201,   4, 144,   9, 162, 255
BF 1220 data 142, 252, 193, 201,   5, 208,  25, 169
FA 1230 data   1, 141,   0, 194, 169,  24, 141,  65
GE 1240 data 193, 169, 105, 141,  66, 193, 169, 176
DI 1250 data 141,  73, 193, 169, 238, 141,  78, 193
OG 1260 data 169,   4, 162,   4, 160,   5,  32, 186
GC 1270 data 255, 169,   0,  32, 189, 255,  32, 192
BB 1280 data 255, 162,   4,  32, 201, 255, 169,  27
DK 1290 data  32, 210, 255, 169,  51,  32, 210, 255
JH 1300 data 169,  16,  32, 210, 255,  32, 172, 193
GJ 1310 data 169,  25, 141, 237, 193, 169,  40, 141
OK 1320 data 238, 193, 173,   0, 194, 208,  21, 173
PL 1330 data 251, 193,  24, 105,   1, 141, 234, 193
GH 1340 data 133, 252, 169,  56, 141, 235, 193, 133
FJ 1350 data 251,  76, 251, 192, 173, 251, 193, 141
PG 1360 data 234, 193, 133, 252, 169,   0, 133, 251
FI 1370 data 141, 235, 193, 160,   0, 162,   8, 173
JF 1380 data   0, 194, 208,  42, 177, 251, 106,  46
HL 1390 data 236, 193, 202, 208, 249, 173, 236, 193
HM 1400 data 153, 239, 193, 200, 192,   8, 208, 229
MK 1410 data 173, 253, 193, 240,   3,  32,   6, 194
IB 1420 data  32, 196, 193, 206, 237, 193, 240,  11
GO 1430 data  32, 158, 193,  76, 251, 192, 177, 251
GP 1440 data  76,  16, 193,  32,  77, 194, 206, 238
EM 1450 data 193, 240,  40,  32, 172, 193, 173, 235
DF 1460 data 193,  56, 233,   8, 141, 235, 193, 133
AA 1470 data 251, 144,   3,  76,  86, 193, 206, 234
IC 1480 data 193, 173, 234, 193, 133, 252, 173, 234
GB 1490 data 193, 133, 252, 169,  25, 141, 237, 193
LK 1500 data  76, 251, 192, 169,  27,  32, 210, 255
GN 1510 data 169,  64,  32, 210, 255, 169,   4,  32
DM 1520 data 195, 255,  32, 204, 255, 169,   0, 141
FE 1530 data 253, 193, 141, 250, 193, 169, 200, 141
PH 1540 data 249, 193, 169,  56, 141,  65, 193, 169
LJ 1550 data 233, 141,  66, 193, 169, 144, 141,  73
IA 1560 data 193, 169, 206, 141,  78, 193,  76,  49
BF 1570 data 234, 104, 104,  76,  99, 193, 165, 251
LC 1580 data  24, 105,  64, 133, 251, 165, 252, 105
NO 1590 data   1, 133, 252,  96, 169,  13,  32, 210
GD 1600 data 255, 169,  10,  32, 210, 255, 162,   0
JH 1610 data 189, 247, 193,  32, 210, 255, 232, 224
LG 1620 data   4, 208, 245,  96, 162,   0, 189, 239
CL 1630 data 193,  77, 252, 193,  32, 210, 255, 173
OC 1640 data 253, 193, 240,   9, 189, 239, 193,  77
ID 1650 data 252, 193,  32, 210, 255, 232, 224,   8
MJ 1660 data 208, 228, 165, 197, 201,  63, 240, 177
LA 1670 data  96,   0,   0,   0,   0,   0,   0,   0
OM 1680 data   0,   0,   0,   0,   0,   0,   0,  27
FK 1690 data  75, 200,   0,  64,   0,   0,   0,   0
OJ 1700 data   0,   1,   8,   0,   0,   0, 160,   0
OM 1710 data 169,   4, 141, 255, 193, 185, 239, 193
BM 1720 data  32,  67, 194,  10, 176,  21,  46, 236
HP 1730 data 193,  24,  46, 236, 193, 206, 255, 193
CC 1740 data 208, 241, 173, 236, 193, 153, 239, 193
PO 1750 data  76,  61, 194,  46, 236, 193,  56,  46
DC 1760 data 236, 193, 206, 255, 193, 208, 220, 173
AF 1770 data 236, 193, 153, 239, 193, 200, 192,   8
BP 1780 data 208, 198,  96, 174, 254, 193, 240,   4
IA 1790 data  10,  10,  10,  10,  96, 173, 253, 193
BG 1800 data 240,  23, 173, 254, 193,  73,   1, 141
FD 1810 data 254, 193, 240,  13, 104, 104, 173, 235
CP 1820 data 193, 133, 251,  32, 172, 193,  76,  86
KK 1830 data 193,  96, 173,   3, 194, 208,  83, 169
OC 1840 data   1, 141, 233, 193, 141,   3, 194, 173
EC 1850 data   0, 221, 141,   4, 194, 173,  24, 208
JL 1860 data 141,   5, 194, 169,  64, 133, 254, 173
CF 1870 data   2, 194, 133, 252, 169,   0, 133, 253
EF 1880 data 173,   1, 194, 133, 251, 160,   0, 177
BI 1890 data 251, 145, 253, 200, 208, 249, 230, 252
PK 1900 data 230, 254, 165, 254, 201,  96, 208, 239
AO 1910 data 173,   0, 221,  41, 252,   9,   2, 141
BI 1920 data   0, 221, 173,  24, 208,  41, 135,   9
ML 1930 data 128, 141,  24, 208,  32,  89, 195,  32
EL 1940 data  71, 195,  76,  50, 192, 173,   3, 194
GN 1950 data 240, 248,  32,  80, 195, 173,   4, 194
NH 1960 data 141,   0, 221, 173,   5, 194, 141,  24
FM 1970 data 208, 169,   0, 141,   3, 194, 141, 233
AC 1980 data 193,  76,  49, 234, 173,   3, 194, 240
HN 1990 data 217, 173, 141,   2,  41,   1, 240,  24
EJ 2000 data 173,   1, 194, 201,   0, 208,  11, 206
NN 2010 data   2, 194, 169, 255, 141,   1, 194,  76
EF 2020 data 131, 194, 206,   1, 194,  76, 131, 194
EA 2030 data 238,   1, 194, 208,   3, 238,   2, 194
IK 2040 data  76, 131, 194, 173,   3, 194, 240, 170
GA 2050 data 173, 141,   2,  41,   1, 240,  20, 173
AC 2060 data   1, 194,  56, 233,  64, 141,   1, 194
```

```
GC  2070 data 173,   2, 194, 233,   1, 141,   2, 194
LF  2080 data  76, 131, 194, 169,  64,  24, 109,   1
ME  2090 data 194, 141,   1, 194, 169,   1, 109,   2
MK  2100 data 194, 141,   2, 194,  76, 131, 194, 173
GM  2110 data  17, 208,   9,  32, 141,  17, 208,  96
EN  2120 data 173,  17, 208,  41, 223, 141,  17, 208
KA  2130 data  96, 160,   0, 162,   0, 169,   1, 157
IH  2140 data   0,  96, 157, 250,  96, 157, 244,  97
KL  2150 data 157, 238,  98, 232, 224, 250, 208, 237
JP  2160 data  96
```

Hires Search and Print: Merlin Format Source Code

Mr. Farrah's program source could have easily been converted to PAL and verifized, but we didn't expect anyone would be entering it by hand - it's here for reference only.

```
*hires screen dump program, vertical screen read tech.
*by jack farrah
*hires picture to be loaded into basic prog. memory
*hires screen located at $4000
*f5 turns on hires screen, space bar returns to text screen
*cursor right advances mem. read by one byte.
*cursor up advances by 320 bytes (1 screen line)
*shift key decrements mem.by same amount
*pictures are printed sideways in dump
*f1 prints small picture.f2 prints enlarged picture
*control prints negative image,shift prints mirror image
*control and shift can be used simultaneously
*constants*
chrout  =  $ffd2        ;kernal print
open    =  $ffc0        ;open file
chkout  =  $ffc9        ;designate output file
setlfs  =  $ffba        ;set file & sec. add.
setnam  =  $ffbd        ;file name
clrchn  =  $ffcc        ;restore default devices
close   =  $ffc3        ;close file
irqvec  =  $314         ;irq vector
bmreg   =  $d011        ;enable hires register
vidbas  =  $d018        ;mem.bank/screen register
bank    =  $dd00        ;set mem. bank register
colmem  =  $6000        ;color mem. for hires
org $c000               ;49152
        sei
        lda  #<new
        sta  irqvec
        lda  #>new
        sta  irqvec+1
        lda  #0
        sta  prgflg     ;busy flag
        sta  picflg     ;clear for small print
        sta  hiflg      ;hires mode flag
        cli
        rts
*new wedged routine to print hires screen
*and search for hires screen
new     jmp  start
*springboards to display and search routines
set1    jmp  sett       ;right cursor routine
setup1  jmp  setup      ;up cursor routine
rstor1  jmp  rstor      ;space bar routine
dsply1  jmp  dsply      ;display hires routine
start   lda  $c5        ;check current key pressed
        cmp  #64        ;64 = no key pressed
        bne  ckmor      ;something, check it out
        lda  #0         ;nothing.keep flg clear
        sta  prgflg
exit    jmp  $ea31      ;normal irq routine
*main loop when keypress detected
ckmor   ldx  prgflg     ;if program flag set, we're
        bne  exit       ;busy now, so exit
        cmp  #6         ;f5 key?
        beq  dsply1     ;yes,branch
        cmp  #60        ;no.space bar?
        beq  rstor1     ;yes
        cmp  #2         ;no.cursor right?
        beq  set1       ;yes
        cmp  #7         ;no.cursor down?
        beq  setup1     ;yes
        cmp  #4         ;no.f1?
        beq  doit       ;was f1.print screen
        cmp  #5         ;no.f3?
        bne  exit       ;not f1 or f3. exit
        lda  #1         ;its f3
        sta  picflg     ;set for large print
        lda  #0
        sta  pass       ;set up for 1st pass
        lda  #144       ;new code for 400
        sta  code+2     ;characters to be sent
        lda  #1         ;to printer for expanded
        sta  code+3     ;print mode
doit    lda  #1         ;set flag to show we're busy
        sta  prgflg
        lda  #0         ;clear reverse flag
        sta  rvflg
        sta  mirflg     ;and clear mirror flag
        lda  $28d       ;check for control & shift
        and  #$05       ;mask out all but sig.bits
        beq  setpr      ;if 0, neither set
        cmp  #4         ;somethings on
        bcc  jstmir     ;i<4,must be bit 0
        ldx  #255       ;>1 so set reverse flag
        stx  rvflg
        cmp  #5         ;is it both rev.&mirror?
        bne  setpr      ;just reverse so branch
jstmir  lda  #1         ;set mirror flag
        sta  mirflg
        lda  #24        ;alter codes in ckclm
        sta  ckclm+14   ;routine to add 8 after
        lda  #105       ;each column instead
        sta  ckclm+15   ;of subtracting
        lda  #176
        sta  subhi-5
        lda  #238
        sta  subhi
*set up printer
setpr   lda  #4         ;file #
        ldx  #4         ;device #
        ldy  #5         ;sec.add.to set interface trans.
        jsr  setlfs
        lda  #0         ;no file name
        jsr  setnam
        jsr  open       ;open the file
        ldx  #4
        jsr  chkout     ;file 4 for output
        lda  #27        ;escape
        jsr  chrout     ;send to printer
        lda  #51        ;sg10 code for line spacing
;to n/144ths inch
        jsr  chrout     ;send it
        lda  #16        ;n = 16.16/144ths = 1/9
        jsr  chrout     ;send it
*routine to get hires screen bytes, convert and print
        jsr  shft       ;send cr,lf and codes
        lda  #25
        sta  rocnt      ;set row counter to 25
        lda  #40
        sta  clmcnt     ;set column counter to 40
        lda  mirflg     ;check if mirror print
        bne  mir        ;if set,skip add 312
        lda  btmp       ;get bitmap address high byte
        clc
        adc  #$01       ;add 256
        sta  savad      ;store it
        sta  $fc        ;and on zero page
        lda  #$38       ;(56)
        sta  savad+1    ;put in low byte storage
        sta  $fb        ;and at zero page pointer
        jmp  strt
mir     lda  btmp       ;put start address of
        sta  savad      ;bitmap in zero page
        sta  $fc        ;pointers
        lda  #0
        sta  $fb
        sta  savad+1
*start address for reading the screen at beginning
*of last screen column
* (312 bytes from start of bitmap)
strt    ldy  #0         ;initialize y to count bytes
gtbyt   ldx  #8         ;x to count rotations
        lda  mirflg     ;check if mirror print
        bne  mkmir      ;if set,skip rotate routine
        lda  ($fb),y    ;get bitmap byte from mem.
rotat   ror             ;put low byte in carry
        rol  hldbyt     ;put carry in low byte holder
        dex             ;lower rotate counter
        bne  rotat      ;do it 8 times
        lda  hldbyt     ;after altering byte
store   sta  bytkp,y    ;store it here till we get 8
        iny             ;raise byte counter
        cpy  #8         ;have we done 8?
        bne  gtbyt      ;no. go back for another
        lda  picflg     ;normal or large print?
        beq  prntit     ;normal
        jsr  exp        ;large.modify bytes
prntit  jsr  prnt       ;yes. print the 8 bytes
        dec  rocnt      ;lower row counter
        beq  ckclm      ;if we've done 25, do
;next column
        jsr  add        ;not finished with column
;so reset pointer for next 8 bytes
        jmp  strt       ;go get the next 8
mkmir   lda  ($fb),y    ;for mirror, just put bytes
        jmp  store      ;unchanged in storage
ckclm   jsr  ckpic      ;normal or large print?
        dec  clmcnt     ;we finished 1 column,so
;lower column counter
        beq  end        ;if we finished 80, close up
        jsr  shft       ;we're not done.do cr,lf
        lda  savad+1    ;get prev. column add. low byte
        sec             ;subtract 8 to back up
        sbc  #8         ;to next column
        sta  savad+1    ;save the new low byte
        sta  $fb        ;and at zero page pointer
        bcc  subhi      ;if cc, need to reduce high byte
        jmp  rest       ;no borrow required
subhi   dec  savad      ;lower high byte
        lda  savad      ;get it
        sta  $fc        ;put in zero pg.pointer
rest    lda  savad      ;get old high byte
        sta  $fc        ;reset zero page
        lda  #25        ;re-initialize row counter
        sta  rocnt
        jmp  strt       ;back for next column
end     lda  #27        ;escape code
        jsr  chrout     ;send it
        lda  #64        ;code to initialize printer
        jsr  chrout     ;send it
        lda  #4
        jsr  close      ;close file 4
        jsr  clrchn     ;reset to default devices
        lda  #0
        sta  picflg     ;reset print size flag
        sta  code+3     ;reset printer codes
        lda  #200       ;if altered by enlarged
        sta  code+2     ;print routine
        lda  #$38       ;restore values in ckclm
        sta  ckclm+14   ;routine to do subtract
        lda  #$e9       ;in case have been altered
        sta  ckclm+15   ;by the mirror routine
        lda  #$90
        sta  subhi-5
        lda  #$ce
        sta  subhi
        jmp  $ea31      ;exit thru normal irq
*routine to handle stop key during printing
stop    pla             ;pull return address
        pla             ;off the stack
        jmp  end        ;and close up shop
*subroutines*
*reset zero page pointer for next lower screen position
*in current column by adding 320
add     lda  $fb        ;get low byte pointer
        clc
        adc  #$40       ;add 64
```

```
        sta   $fb          ;put new value back
        lda   $fc          ;get high byte
        adc   #$01         ;add 1(value of 256)
        sta   $fc          ;store it back
        rts                ;return
•send code to printer to do carriage return, line feed
•and send printer codes for graphics.200 bytes to follow
•400 bytes if enlarged print mode chosen
shft    lda   #$0d         ;13.carriage return
        jsr   chrout       ;send it
        lda   #10          ;line feed code
        jsr   chrout       ;send it
        ldx   #0           ;x as index
mrcd    lda   code,x       ;get code
        jsr   chrout       ;send it
        inx                ;raise index
        cpx   #4           ;finished all 4?
        bne   mrcd         ;no. do more
        rts                ;yes.return
•routine to print the 8 converted bytes
prnt    ldx   #0           ;x counts bytes
nxtbyt  lda   bytkp,x      ;get saved/conv.byte
        eor   rvflg        ;switch bits if reverse
        jsr   chrout       ;send it
        lda   picflg       ;normal or expanded?
        beq   incx         ;normal, branch
        lda   bytkp,x      ;expanded.print byte
        eor   rvflg        ;twice
        jsr   chrout
incx    inx                ;raise counter
        cpx   #8           ;done all 8?
        bne   nxtbyt       ;no.get another
        lda   $c5          ;check if key hit
        cmp   #63          ;stop key
        beq   stop         ;yes, so close up
        rts                ;yes.return
prgflg  ds 1               ;prg. in progress flag
savad   ds 2               ;holder for current sc. column
;add.(hi/lo order)
hldbyt  ds 1               ;holder for revised order byte
rocnt   ds 1               ;screen row counter
clmcnt  ds 1               ;screen column counter
bytkp   ds 8               ;storage for 8 convert. bytes
code    dfb 27,75,200,0
•codes-escape,set for bitmap mode,# bytes coming(l/h)
btmp    hex 40             ;high byte of bitmap screen
rvflg   ds 1               ;reverse print flag.clear,
;no reverse.set,reverse.
picflg  ds 1               ;clear for normal,set
;for expanded print
pass    ds 1               ;flag to show which pass thru
•;screen column we're on
bitcnt  ds 1               ;counter for bits in expanded
;print byte conversion
mirflg  ds 1               ;flag for mirror print
srclo   hex 01             ;low byte source screen
srchi   hex 08             ;high byte same
hiflg   ds 1               ;hires mode flag
bnkstor ds 1               ;text sc. reg. save
bastor  ds 1               ;same
•conversion routine to double scale high or low
•nybble of byte for expanded print
exp     ldy   #0           ;initialize index into table
get     lda   #4
        sta   bitcnt       ;set counter for 4 bits
        lda   bytkp,y      ;get byte
        jsr   pascnt       ;which pass?
nxtbt   asl                ;shift bit 7 into carry
        bcs   set          ;bit was on
        rol   hldbyt       ;carry into low bit holder
        clc
        rol   hldbyt       ;twice
        dec   bitcnt       ;lower bit counter
        bne   nxtbt        ;do next bit(a still has byte)
        lda   hldbyt       ;we've done 4
        sta   bytkp,y      ;save it
        jmp   reset        ;update counter
set     rol   hldbyt       ;bit was set.put in holder
        sec                ;set carry to put 2 set
        rol   hldbyt       ;bits in holder
        dec   bitcnt       ;lower bit counter
        bne   nxtbt        ;if more to do, get it
        lda   hldbyt       ;done all 4
        sta   bytkp,y      ;save expanded nybble
reset   iny                ;byte done,reset for next
        cpy   #8           ;finished 8?
        bne   get          ;no, get another
        rts                ;yes,return
•adjust byte for pass 1 or 2
pascnt  ldx   pass         ;1st or 2nd pass?
        beq   retrn        ;1st.no change needed
        asl                ;2nd pass. move low
        asl                ;nybble to high
        asl
        asl
retrn   rts
•after doing a column, check if in expanded mode
•and if so, adjust pass counter
ckpic   lda   picflg       ;which mode?
        beq   nochg        ;if clear, just return
        lda   pass         ;get pass value
        eor   #01          ;switch the bit. if it was
;0 it's now 1.if it was 1,
;its now 0
        sta   pass         ;put new value back
        beq   nochg        ;if 0,we just did 2nd pass
        pla                ;pull return address
        pla                ;from stack
        lda   savad + 1    ;restore low byte clm point.
        sta   $fb
        jsr   shft         ;do cr,lf and codes
        jmp   rest         ;restore high pointer,reset
;row counter and redo last clm.
nochg   rts
•routine to display bitmap
dsply   lda   hiflg        ;are we in hires?
        bne   sb           ;yes, so exit
        lda   #1           ;no.set flag to show
        sta   prgflg       ;we're busy and
        sta   hiflg        ;in hires mode
        lda   bank         ;get text screen vic
        sta   bnkstor      ;register and save it
        lda   vidbas       ;same for this register
        sta   bastor
•entry point from cursor routines
enter   lda   #$40         ;high byte of display screen
        sta   $fe          ;into zero page
        lda   srchi        ;source block high byte
        sta   $fc          ;into zero page
        lda   #00          ;low byte display screen
        sta   $fd
        lda   srclo        ;low byte source block
        sta   $fb
        ldy   #0           ;initialize y
•copy 8000 bytes from source to display
lup     lda   ($fb),y      ;get source byte
        sta   ($fd),y      ;store in display
        iny                ;raise index
        bne   lup          ;get 256 bytes
        inc   $fc          ;after 256, raise high byte
        inc   $fe          ;source and display
        lda   $fe          ;get value in display
        cmp   #$60         ;finished when = $60
        bne   lup          ;not done, get more
        lda   bank         ;get register
        and   #$fc         ;mask out bits 0 & 1
        ora   #$02         ;set bit 1
        sta   bank         ;set to bank 1($4000-7fff)
        lda   vidbas       ;get register
        and   #$87         ;clear bits 3,4,5 & 6
        ora   #$80         ;set bit 7
        sta   vidbas       ;sc.mem.at $6000, hires in
;lower half of bank
        jsr   filcol       ;set color mem to black/white
        jsr   biton        ;turn on the hires screen
sb      jmp   exit         ;exit thru normal irq
•routine to reset to text screen in response to space bar
rstor   lda   hiflg        ;are we in hires?
        beq   sb           ;no, so exit
        jsr   bitof        ;yes. turn off bitmap
        lda   bnkstor      ;get text screen register
        sta   bank         ;restore it
        lda   bastor       ;get 2nd register
        sta   vidbas       ;restore it
        lda   #0           ;clear the busy and
        sta   hiflg        ;hires flags
        sta   prgflg
        jmp   $ea31        ;exit thru normal irq
•routine to increment/decrement source screen start
•by one byte in response to right cursor
sett    lda   hiflg        ;are we in hires?
        beq   sb           ;no, so exit
        lda   $028d        ;yes.shift key pressed?
        and   #$01         ;mask all but bit 0
        beq   adlo         ;clear so increment
        lda   srclo        ;set so subtract
        cmp   #0           ;is source low byte 0?
        bne   sb2          ;no, so lower low byte by 1
        dec   srchi        ;yes.decrement hi byte
        lda   #255         ;and add 255 to low byte
        sta   srclo        ;for effective 1 byte reduction
        jmp   enter        ;go update screen
sb2     dec   srclo        ;subtract 1 from low byte source
        jmp   enter        ;update screen
adlo    inc   srclo        ;add 1 to source low byte
        bne   sb1          ;if new value not 0
        inc   srchi        ;if low byte rolled over
;to 0, increment high byte
sb1     jmp   enter        ;update screen
•routine to increment/decrement source start by 320 bytes
setup   lda   hiflg        ;are we in hires?
        beq   sb           ;no.exit
        lda   $028d        ;yes.shift pressed?
        and   #$01         ;mask all but bit 0
        beq   addit        ;if 0, no shift
        lda   srclo        ;it was set. subtract
        sec                ;get low byte source
        sbc   #64          ;subtract 64
        sta   srclo        ;store new value
        lda   srchi        ;get high byte
        sbc   #1           ;subtract 1 (=256)
        sta   srchi        ;store new value
        jmp   enter        ;update screen
addit   lda   #64          ;add 64 to source low byte
        clc                ;and 1 (256) to source
        adc   srclo        ;high byte to raise
        sta   srclo        ;pointer by 1 screen line
        lda   #1
        adc   srchi
        sta   srchi        ;store new values
        jmp   enter        ;update screen
•routine to turn on hires screen
biton   lda   bmreg        ;get enable register
        ora   #$20         ;set bit 5
        sta   bmreg        ;store new value back
        rts                ;return
•routine to turn hires screen off
bitof   lda   bmreg        ;get enable register
        and   #$df         ;clear bit 5
        sta   bmreg        ;store new value
        rts                ;return
•routine to set hires color memory
filcol  ldy   #0           ;initialize y
        ldx   #0           ;and x
colup   lda   #$01         ;0 = color code black, 1 = white
        sta   colmem,x     ;put in all 1000 bytes
        sta   colmem + 250,x
        sta   colmem + 500,x
        sta   colmem + 750,x
        inx                ;raise index
        cpx   #250         ;do 250 times
        bne   colup        ;not done.do more
        rts                ;return
```

Commodore 128 High-Res Graphics

Paul T. Durrant
Long Prairie, Minnesota

Hi–Res Graphics On The Commodore 128's 80 Column Screen Using BASIC 7.0 Commands.

The Commodore 128 computer includes some very useful graphics commands in its BASIC implementation. It also includes two video chips: a 40 column chip (the VIC)---as in the Commodore 64---and an 80 column chip (the VDC). The VDC can display a single color (plus background) High–Resolution graphics image which is 640 dots wide by 200 dots tall: twice the horizontal resolution of the VIC screen.

Unfortunately, the Hi–Res graphics commands included in BASIC 7.0 do not support the VDC. Using the accompanying program, you can create Hi–Res graphics on the 40 column screen and transfer them to either side of the 80 column screen (and back again). With this technique, a 640 dot wide Hi–Res image may be created in two halves, using BASIC 7.0 commands.

Even if you aren't interested in having 80 column Hi–Res graphics, the program may be useful to you. The VDC has access to 16K of its own dedicated memory---outside of the normal address space. Thus you may store and retrieve two 40 column Hi–Res screens without any loss of main memory.

Use Of The Program

The SYS command in BASIC 7.0 allows you to directly pass a value to the processor's accumulator (and other registers).

SYS <address>,<value>

. . .will start the machine language program at <address> with <value> in the accumulator. This program takes advantage of this expanded capability of the SYS command. As presented, the routine begins at location 2816 ($b00)---in the 128's cassette buffer.

As noted, the 80 column Hi–Res screen can display two 40 column screen's worth of information---one on the left side, one on the right. This program allows you to select which half of the 80 column screen to use. Not only can you transfer an image to the 80 column screen from the 40 column screen, you also may transfer one **back again**. Four different combinations are possible:

SYS 2816,0 – transfers the 40 column screen to the left side of the 80 column screen.

SYS 2816,1 – transfers the 40 column screen to the right side of the 80 column screen.

SYS 2816,2 – transfers the left side of the 80 column screen to the 40 column screen.

SYS 2816,3 – transfers the right side of the 80 column screen to the 40 column screen.

In addition, SYS 3023 can be used to clear and restore the VDC screen to text mode.
You may easily program the Function Keys to execute these commands.

How It Works:

Before discussing the program, it's necessary to review a little of how Hi–Res graphics images are created by the two chips. A single text **character** is composed of 64 bits in an 8 by 8 matrix. Each time the monitor scans a single screen line, it picks up one byte (8 bits) and either turns the "light beam" on or off for each bit. Data for a Hi–Res image in the 40 column mode start at address $2000. Location $2001 refers to the **second byte** of the "character" in the upper left corner (which is on the second screen line). Location $2002 refers to the third byte, etc. After creating the dots for the top row of the **first character**, the next set of dots on the monitor must come from the top byte of the **second character**. So the VIC reads every eighth byte as it goes across the screen, moves down one byte from where it started, and then displays every eighth byte again, until the whole first row of "characters" is displayed. This arrangement is illustrated below:

```
$2000   $2008   $2010  . . .  $2138
$2001   $2009   $2011  . . .  $2139
$2002   $200a   $2012  . . .  $213a
  .       .       .    . . .    .
  .       .       .    . . .    .
  .       .       .    . . .    .
$2007   $200f   $2017  . . .  $213f
$2140   $2148   $2150  . . .  $2278
  .       .       .    . . .    .
  .       .       .    . . .    .
  .       .       .    . . .    .
```

Anyone who has dabbled with Hi–Res graphics on the Commodore 64 (which uses the same procedure) knows that this is a rather clumsy technique.

The new VDC uses a simpler method. When in Hi–Res mode, location $0000 of the VDC's dedicated memory refers to the first byte on the top screen line. Location $0001 refers to the next byte on the top screen line, etc. As you can see, this is a much simpler arrangement:

```
$0000   $0001   $0002  . . .  $0049
$0050   $0051   $0052  . . .  $00a0
  .       .       .    . . .    .
  .       .       .    . . .    .
  .       .       .    . . .    .
```

At the beginning of the program the accumulator is checked to see if it contains one of the four defined instructions. If so, the VDC is placed in graphics mode, the instruction is saved, and pointers are initialized. ROW.COUNT is initialized to 25, 40 " characters " worth of data are moved, pointers are updated and ROW.COUNT is decremented. The program is finished when ROW.COUNT gets to zero. The accompanying annotated disassembly of the machine language code tells it all.

Much of the code could be written in one large routine, but I've chosen to break it into several smaller ones. Some of the subroutines are only called once from a single higher routine, but the resulting code is easier to follow.

Even though this is a machine language program, it takes a second or so for a transfer between screens to be completed. This is caused by the different arrangements of Hi–Res data in memory, the indirect addressing required for the VDC's 16K of dedicated memory, and the fact that 8K of data is being moved each time.

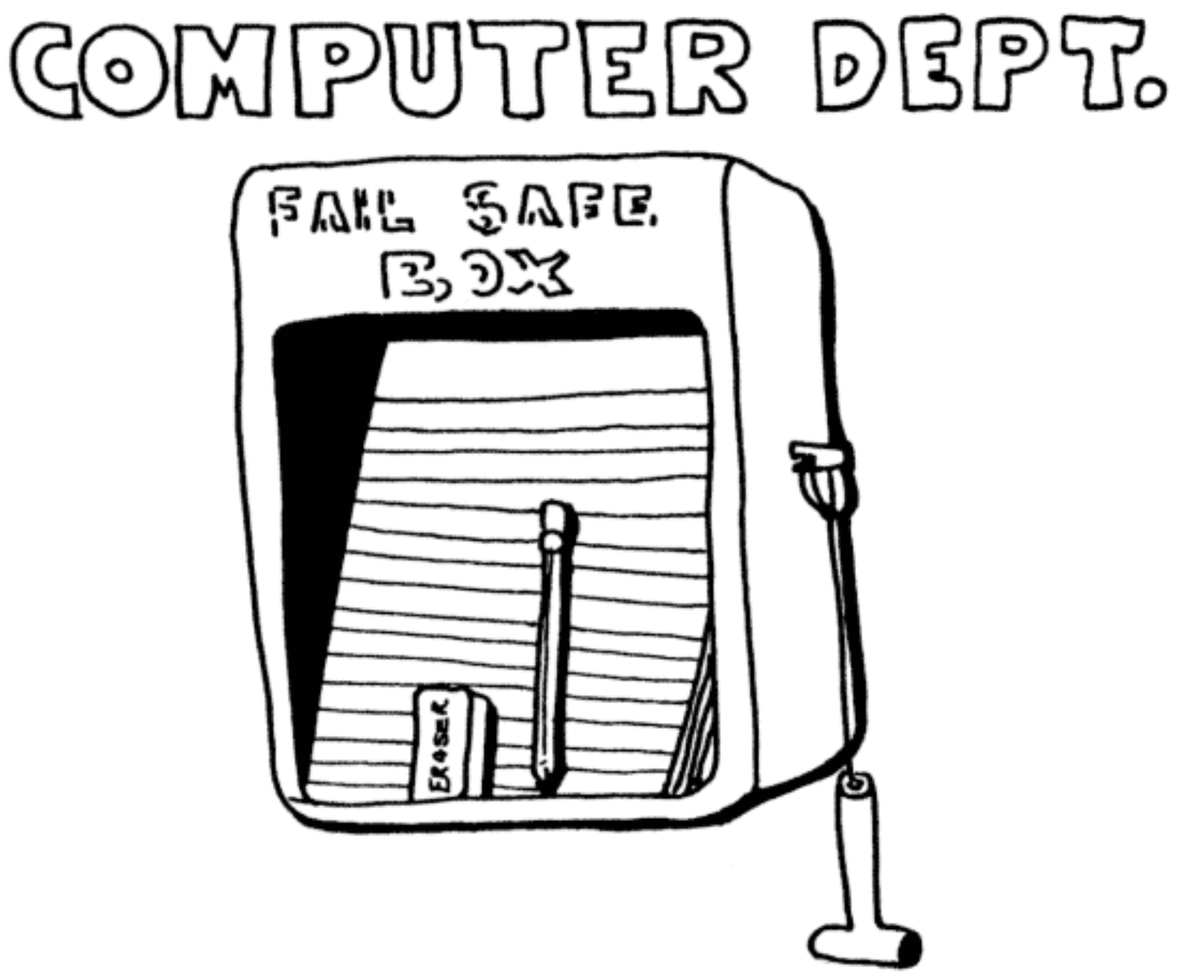
```
;Entry/initialization:
00b00 c9 04     cmp #$04
00b02 90 01     bcc $0b05       ;Branch if (acc) < 4
00b04 60        rts             ;else ignore "illegal" com-
                                mand.
00b05 48        pha
00b06 20 c7 0b  jsr $0bc7       ;Set 80 col screen to graphics.
00b09 68        pla
00b0a 8d dc 0b  sta $0be6       ;Save TRANSFER.DIREC-
                                TION.
00b0d 29 01     and #$01        ;If (acc) = 1 now, refers to
00b0f f0 02     beq $0b13       ;right side of 80 col screen,
00b11 a9 28     lda #$28        ;so start at middle.
00b13 85 e0     sta $e0         ;(Acc) = 0 or 40 now.
00b15 a0 00     ldy #$00
00b17 84 e1     sty $e1         ;$e0/e1 are the 80.COL.PTR.
00b19 84 fb     sty $fb         ;Low byte of 40.COL.PTR
00b1b 8c de 0b  sty $0be8       ;BEGINning of 40 col ROW.
00b1e a9 20     lda #$20
00b20 85 fc     sta $fc         ;High byte of 40.COL.PTR.
00b22 8d df 0b  sta $0be9       ;Ditto, for BEGIN of 40 col
                                ROW.

;Move screen:
00b25 a9 19     lda #$19        ;25 screen rows.
00b27 8d dd 0b  sta $0be7       ;ROW.COUNTER.
00b2a 20 43 0b  jsr $0b43       ;Do "move 40 characters".
00b2d e6 fb     inc $fb         ;Set 40.COL.PTR to
00b2f a5 fb     lda $fb         ;next row.
00b31 8d de 0b  sta $0be8       ;Save it in BEGIN.ROW, too.
00b34 d0 02     bne $0b38
00b36 e6 fc     inc $fc         ;Adjust high bytes of
00b38 a5 fc     lda $fc         ;both if necessary.
00b3a 8d df 0b  sta $0be9
00b3d ce dd 0b  dec $0be7       ;ROW.COUNTER.
00b40 d0 e8     bne $0b2a       ;Repeat if not done,
00b42 60        rts             ;else rts.

;Move 40 characters:
00b43 a9 08     lda #$08        ;Each "character" has 8
                                bytes.
00b45 8d e0 0b  sta $0bea       ;So count BYTE.ROW down.
00b48 20 59 0b  jsr $0b59       ;jsr "move 40 bytes",
00b4b 20 98 0b  jsr $0b98       ;and move to next 80 col row.
00b4e ce e0 0b  dec $0bea       ;If done 8 times,
00b51 d0 01     bne $0b54       ;then
00b53 60        rts             ;rts.
00b54 20 a9 0b  jsr $0ba9       ;Else move to next 40 col row
00b57 d0 ef     bne $0b48       ;and branch always.

;Move 40 bytes:
00b59 ad dc 0b  lda $0be6       ;TRANSFER.DIRECTION.
00b5c c9 02     cmp #$02        ;If <2 then
00b5e 90 12     bcc $0b72       ;branch to move from 40 to 80.
00b60 20 c0 0b  jsr $0bc0       ;Else move from 80 to 40.
00b63 8c e1 0b  sty $0beb       ;Save (y) as ptr to col on 80
00b66 a0 00     ldy #$00        ;col screen and use for indirect
00b68 91 fb     sta ($fb),y     ;store to 40.
00b6a ac e1 0b  ldy $0beb       ;Restore (y) and
00b6d 20 84 0b  jsr $0b84       ;increment pointers.
00b70 d0 ee     bne $0b60       ;Branch always.
```

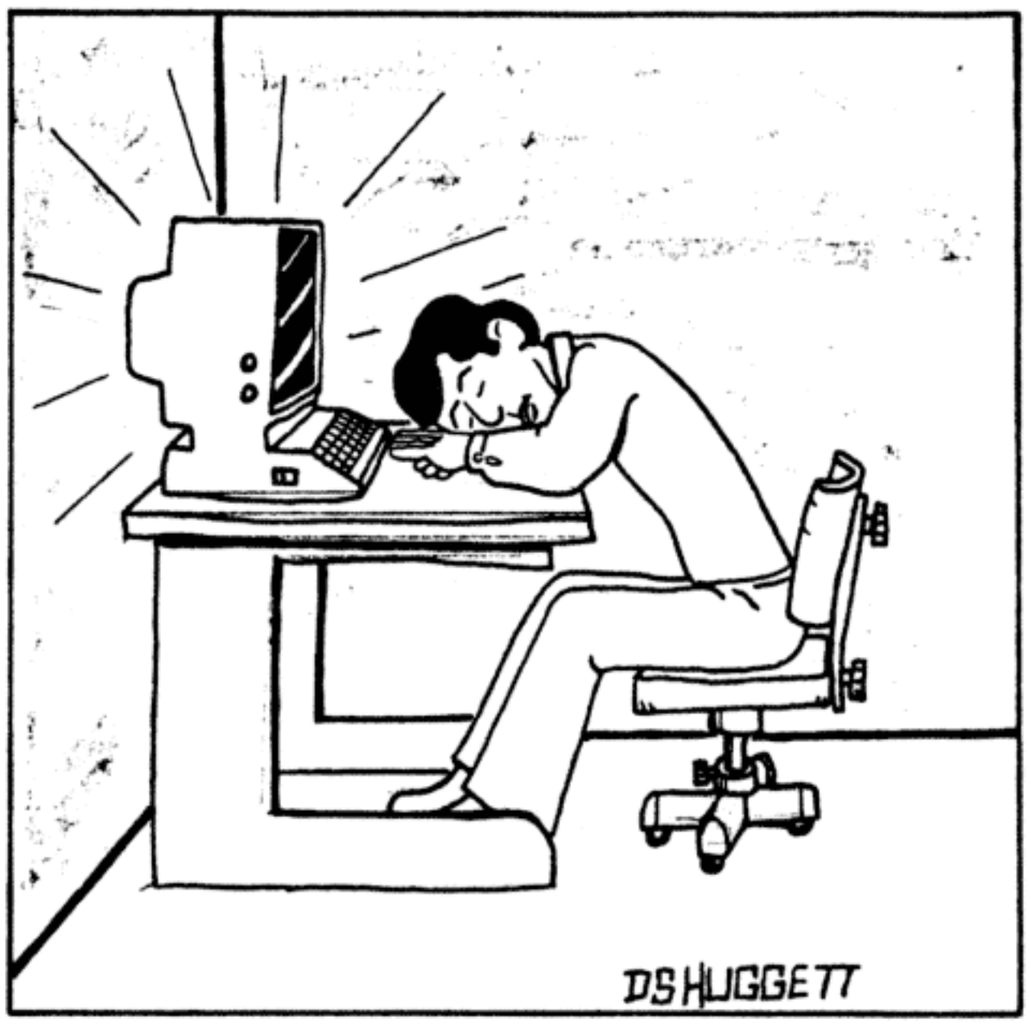
```
;Move a byte from 40 to 80:
00b72 8c e1 0b  sty  $0beb   ;Save (y) as ptr to col on 80
00b75 a0 00     ldy  #$00    ;screen.
00b77 b1 fb     lda  ($fb),y ;Get byte from 40 col
00b79 ac e1 0b  ldy  $0beb   ;screen
00b7c 20 bc 0b  jsr  $0bbc   ;and store in 80.
00b7f 20 84 0b  jsr  $0b84   ;Increment pointers and
00b82 d0 ee     bne $0b72    ;branch always.


;Increment 40.COL.PTR
00b84 c8        iny
00b85 c0 28     cpy #$28     ;Have 40 bytes been moved?
00b87 90 03     bcc $0b8c    ;Branch if not.
00b89 68        pla          ;Else pull return address off
00b8a 68        pla          ;stack and return to
00b8b 60        rts          ;"move 40 characters".
00b8c 18        clc          ;Add
00b8d a9 08     lda #$08     ;8 to 40.COL.POINTER to get
00b8f 65 fb     adc $fb      ;next byte this screen line.
00b91 85 fb     sta $fb
00b93 90 02     bcc $0b97
00b95 e6 fc     inc $fc
00b97 60        rts


;Next 80 col row:
00b98 a0 00     ldy  #$00    ;Start with first byte.
00b9a 18        clc
00b9b a9 50     lda  #$50    ;Add 80 to 80 col ptr.
00b9d 65 e0     adc  $e0
00b9f 85 e0     sta  $e0
00ba1 90 02     bcc  $0ba5
00ba3 e6 e1     inc  $e1
00ba5 20 e6 cd  jsr  $cde6   ;ROM routine which sets
00ba8 60        rts          ;location in VDC's memory.


;Next 40 col row:
00ba9 ee de 0b  inc  $0be8   ;BEGIN.ROW
00bac ad de 0b  lda  $0be8   ;$0be8 points to beginning of a
00baf 85 fb     sta  $fb     ;row of bytes in 40 col screen.
00bb1 d0 03     bne  $0bb6   ;This routine is called after
00bb3 ee df 0b  inc  $0be9   ;40 bytes have been moved, to
00bb6 ad df 0b  lda  $0be9   ;set 40.COL.PTR to
00bb9 85 fc     sta  $fc     ;the next row of bytes.
00bbb 60        rts
```

```
;Store a byte in VDC's memory:
00bbc 48        pha          ;Set VDC memory location
00bbd 4c 54 cc  jmp  $cc54   ;and store a byte there.


;Get a byte from VDC's memory:
00bc0 20 e6 cd  jsr  $cde6   ;Select VDC memory location
                             and
00bc3 20 d8 cd  jsr  $cdd8   ;read a byte from there.
00bc6 60        rts


;Turn on VDC's graphics mode:
00bc7 a2 19     ldx  #$19
00bc9 a9 80     lda  #$80
00bcb 20 cc cd  jsr  $cdcc
00bce 60        rts


;Return VDC to text mode:
00bcf a2 19     ldx  #$19    ;Text mode.
00bd1 a9 40     lda  #$40
00bd3 20 cb 0b  jsr  $0bcb
00bd6 a5 d7     lda  $d7     ;Cursor in 80 col screen?
00bd8 30 03     bmi  $0bdd   ;Branch if so,
00bda 20 2c cd  jsr  $cd2c   ;else switch screens.
00bdd 20 42 c1  jsr  $c142   ;Clear screen.
00be0 20 2c cd  jsr  $cd2c   ;Switch to 40 col screen.
00be3 4c 0c ce  jmp  $ce0c   ;Restore VDC character set.
```

Computoons

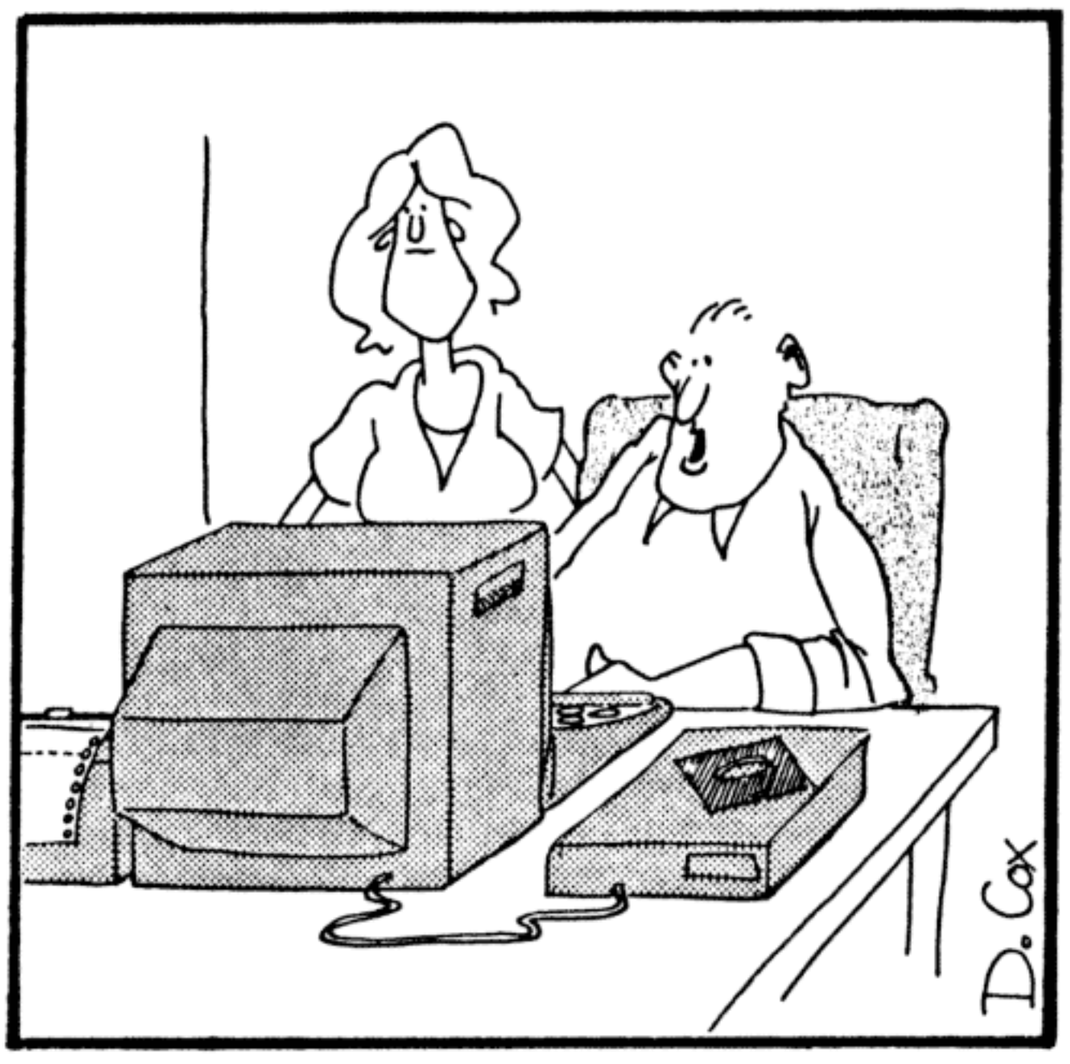
"RUNNING A PROGRAM"





"This darn human is down more than it's up!"

"You mean there's a fruit name after a computer?!"



"According to this Home Finance program, we're broke because we bought this computer."

News BRK

Submitting NEWS BRK Press Releases

If you have a press release which you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way Applicable to Commodore equipment. News of events such as computer shows should be received at least 6 months in advance.

Transactor News

Transactor Mail Order News

Our mail-order department is expanding nicely, but our mail-order card isn't. Seems we just can't find any more room to put more text without making it so small that you can't read it. So, if you're using the card to order, we suggest you pull it out and cross-reference with the list below for more details.

■ Inner Space Anthology $14.95
This is our ever popular Complete Commodore Inner Space Anthology. Even after a year, we still get inquiries about its contents. Briefly, The Anthology is a reference book – it has no "reading" material (ie. "paragraphs"). In 122 compact pages, there are memory maps for 5 CBM computers, 3 Disk Drives, and maps of COMAL; summaries of BASIC commands, Assembler and MLM commands, and Wordprocessor and Spreadsheet commands. Machine Language codes and modes are summarized, as well as entry points to ROM routines. There are sections on Music, Graphics, Network and BBS phone numbers, Computer Clubs, Hardware, unit-to-unit conversions, plus much more. . . about 2.5 million characters total!

■ The Toolbox (PAL and POWER) $79.95
PAL and POWER from Pro-Line are two of the most popular programs for the Commodore 64. PAL is an easy-to-use assembler (most assembler listings in The Transactor are in PAL format), and POWER is a programmer's aid package that adds editing features and useful commands to the programming environment. They come with two nice manuals, and our price is $50 less than suggested retail!

Amiga RAM Expansion by Comspec

■ AX1000 Amiga 1 MEG RAM Box $729.00 (+$100 S&H) U.S.,
$1035.00 (+$25 S&H) Cdn
■ AX2000 Amiga 2 MEG RAM Box $899.00 (+$100 S&H) U.S.,
$1276.00 (+$25 S&H) Cdn
The AX2000 adds 2 Megabytes of "fast" RAM to the Amiga, allowing more tasks to run in the system at once, or for use as a fast RAM-drive. The unit plugs into the expansion connector on the side of the Amiga and duplicates the connector for other devices to plug into. Up to two RAM boards may be plugged in together (limited by the Amiga'a power supply), adding 4 Megabytes. The box has "auto-config", so with Kickstart 1.2 the RAM will automatically be added to the system when it is booted. If you are using Kickstart 1.0 or 1.1 (no auto-config), you can use the program included with the AX2000 to add the memory to the system, and change your startup-sequence to automatically add the memory on power-up. Standard expansion bus architecture was used in the design of the AX2000, ensuring compatability with all peripherals and operating system releases. The unobtrusive steel box is the same height and colour as the Amiga, and snugs up to the side without taking up much extra space. The unit is built tough and comes with a 1 year manufacturer warranty.

This seems to be the most highly-recommended Amiga RAM board, and the first one to actually be available, so we're selling it here at The Transactor. You can order the AX2000 or the 1-Meg AX1000 from the subscription form in this issue. Shipping and Handling to the U.S.A. is via courrier and includes all customs clearance, or you can opt to clear shipments yourself and have it shipped "collect". For dealer information, contact:

Comspec Communications Inc.
153 Bridgeland Avenue
Toronto, Ont.
M6A 2Y6 (416) 787-0617

Paperback Writer now "Pocket Writer"

To avoid confusion with products by Paperback Software International of California manufactured for the IBM market, Digital Solutions has changed the name of their Paperback series of software to Pocket Writer, Pocket Planner, Pocket Filer, and Pocket Dictionary. The new packaging will commense when current stock runs out, but the software inside will be identical.

■ Pocket Writer C64 $39.95 US, $49.95 Cdn
■ Pocket Planner C64 $39.95 US, $49.95 Cdn
■ Pocket Filer C64 $39.95 US, $49.95 Cdn
■ Pocket Writer C128 $49.95 US, $69.95 Cdn
■ Pocket Planner C128 $49.95 US, $69.95 Cdn
■ Pocket Filer C128 $49.95 US, $69.95 Cdn
■ Pocket Dictionary $14.95 US, $19.95 Cdn
In our opinion, the Pocket packages from Digital Solutions are the best you can get on their own – the fact that they work with each other makes them even better. Planner and Filer data can be loaded into the Writer, Writer text can be sent to the Filer, and etcetera. The Dictionary spell checker works with both versions of the Writer.

■ The GLINK C64 to IEEE Interface $49.95
The GLINK plugs into the cartridge port, but doesn't extend the port for more cartridges (for that you'll need a "motherboard" of some kind). The other side of the GLINK is an IEEE card-edge suitable for a PET-IEEE cable. From there, any IEEE device can be accessed including disk drives, modems, printers, etc. The GLINK is "transparent" – that means it won't interfere with programs, except those that rely on the serial routines which it replaces (ie. programs with built-in "fastloaders" for the 1541 won't like the presence of the GLINK). It has no manual (aside from one page of installation instructions) because it alters nothing and leaves everything unchanged! An on-board switch allows you to select Serial or IEEE. GLINK works with both the C64 and the C128 in 64 mode, but not on the VIC 20.

■ The TransBASIC Disk $9.95
This is the complete collection of every TransBASIC module ever published up to Volume 7, Issue 01. There are over 120 commands at your disposal. You pick the ones you want to use, and in any combination! It's so simple that a summary of instructions fits right on the disk label. The manual describes each of the commands, plus how to write your own commands.

■ Super Kit 1541 $29.95 US, $39.95 Cdn
Super Kit is, quite simply, the best disk file utility there is. No more losing those valuable copy-protected originals (like what's happened to me twice too many times). So far we've shipped over 100 Super Kits and orders continue to pour in.

Gnome Speed Compiler = SM Compiler

■ Gnome Speed Compiler $59.95 US, $69.95 Cdn
In last issue's NEWS BRK section, the SM compiler was incorrectly listed at $39.95 U.S. The SM compiler's real name is "Gnome Speed" as introduced in the last page ad, and sells for the price listed in the ad and on the subscription card: $59.95 U.S. This compiler is for BASIC 7.0 on the Commodore 128.

■ Gnome Kit Utility $39.95 US, $49.95 Cdn
Gnome Kit is a Commodore 128 utility with enhancements for the BASIC editor (like Trace, Find, Renumber, Delete, Auto, etc.) as well as enhanced monitor commands, and floppy disk monitor functions.

Transactor Disks, Transactor Back Issues, and Microfiche

All issues of The Transactor from Volume 4 Issue 01 forward are now available on microfiche. According to Computrex, our fiche manufacturer, the strips are the "popular 98 page size", so they should be compatible with every fiche reader. Some issue are ONLY available on microfiche – these are marked "MF only". The other issues are available in both paper and fiche. Don't check both boxes for these unless you want both the paper version and the microfiche slice for the same issue.

To keep things simple, the price of Transactor Microfiche is the same as magazines, with one exception. A single back issue will be $4.50 and subscriptions are $15.00. The exception? A complete set of 18 (Volumes 4, 5, and 6) will cost just $39.95!

This list also shows the "themes" of each issue. "Theme issues" didn't start until Volume 5, Issue 01.

- ■ Vol. 4, Issue 01 (■ Disk 1)
- ■ Vol. 4, Issue 02 (■ Disk 1)
- ■ Vol. 4, Issue 03 (■ Disk 1)
- ■ Vol. 4, Issue 04 – MF only (■ Disk 1)
- ■ Vol. 4, Issue 05 – MF only (■ Disk 1)
- ■ Vol. 4, Issue 06 – MF only (■ Disk 1)
- ■ Vol. 5, Issue 01 – Sound and Graphics (■ Disk 2)
- ■ Vol. 5, Issue 02 – Transition to Machine Language (■ Disk 2)
- ■ Vol. 5, Issue 03 – Piracy and Protection – MF only (■ Disk 2)
- ■ Vol. 5, Issue 04 – Business & Education – MF only (■ Disk 3)
- ■ Vol. 5, Issue 05 – Hardware & Peripherals (■ Disk 4)
- ■ Vol. 5, Issue 06 – Aids & Utilities (■ Disk 5)
- ■ Vol. 6, Issue 01 – More Aids & Utilities (■ Disk 6)
- ■ Vol. 6, Issue 02 – Networking & Communications (■ Disk 7)
- ■ Vol. 6, Issue 03 – The Languages (■ Disk 8)
- ■ Vol. 6, Issue 04 – Implementing The Sciences (■ Disk 9)
- ■ Vol. 6, Issue 05 – Hardware & Software Interfacing (■ Disk 10)
- ■ Vol. 6, Issue 06 – Real Life Applications (■ Disk 11)
- ■ Vol. 7, Issue 01 – ROM / Kernel Routines (■ Disk 12)
- ■ Vol. 7, Issue 02 – Games From The Inside Out (■ Disk 13)

Notes: The Transactor Disk #1 contains all program from Volume 4, and Disk #2 contains all programs from Volume 5, Issues 1–3. Afterwards there is a separate disk for each issue. Disk 8 from The Languages Issue contains COMAL 0.14, a soft-loaded, slightly scaled down version of the COMAL 2.0 cartridge. And Volume 6, Issue 05 published the directories for Transactor Disks 1 to 9.

No Sales Tax on Books

Residents of Ontario need not add the 7% sales tax for the Inner Space Anthology and Jim Butterfield's 1986 Diary as indicated on the subscription card.

Sending Cheques For Transactor Products

If you wish to send a cheque with your subscription/order form, or you wish to conceal your credit card number, you can use an envelope and tape it to the back of the subscription card. The post office has threatened to charge us extra for sloppy business reply mail so please try to use an envelope that is smaller than the card. Can't find one? Just trim the end off the envelope and tape along that edge when fixing it to the card.

The Transactor Communications Disk

We're still working on the "Transactor Communications Disk". Of course the Viewtron software is no longer being considered. And the fact that we're now working on a new home for our on-line services means that this project had to "pushed ahead". Time for the project has been "slotted" and we hope to have it near completion either before or slightly after release of the next issue. When finished it could host as many as 15 different modem programs and may even require two diskettes. We also plan an "all-in-one" manual to go with it so you'll never be without a telecommunications program for virtually any host computer and protocol. But it's not ready yet so don't send any orders. More next issue.

Demise of Viewtron

Yes, it really happened: newspaper chain Knight-Ridder pulled the plug on the Viewtron online service that we've been raving about so much lately. And just when the Transactor section was doing so well (Heavy sigh). Perhaps some of the other services will take Viewtrons place in the future.

Quantum Link and Timeline

We're currently investigating Quantum Link and Timeline (a Montreal based multi-user online service) as the new homes for The Transactor Online Data Service. At the moment, however, neither is readily available to all our subscribers simultaneously. Timeline access is through DataPac, a packet switching service that seems to be only in Canada. Quantum Link can be accessed via Uninet and Telenet which don't yet extend beyond the U.S. TymNet has recently installed nodes in 5 or 6 major Canadian centers, but does not carry the Quantum Link service. However, remote areas would still need to call long distance. The ideal solution would be to have Quantum Link on DataPac which they are reportedly close to completing. Unfortunately we can't offer any more definite plans, but can say that it is on top of the priority list for things to do between now and next issue.

Using Transactor Programs In Proprietary Software

Our policy concerning the use of our routines in your own software, commercial or otherwise, is this: you have our blessings. If you can find a use for something that we've published, well, that's what we're here for. We won't demand royalties or even a free copy, but we'd be happy if you gave the Transactor credit in some way, as well as the author of your assistance.

Industry News

1986 Midwest Commodore Conference/Expo

The 1986 Midwest Commodore Conference/Expo is to be held on August 9th and 10th, 1986 at the Holiday Inn Convention Centre, 72nd and Grover in Omaha, Nebraska. There will be over 30 workshops and both local and national vendor support. Speakers include Jim Butterfield from Toronto, Ontario, Dr. Richard Immers, the co-author of "Inside Commodore DOS" from Detroit, Michigan, Valerie Kramer, a computer language expert from Los Angelas, California, Dr. James Alley, Art Professor (Amiga) from the Savannah College Of Arts, Savanaugh, Georgia, and Pete Baczor, the Commodore Users Group Coordinator. It looks to be one of the biggest Commodore shows in the country. For more information, contact:

The Greater Omaha Commodore Users Group
P.O. Box 241155
Omaha, Nebraska
Attention: Tim Trabold

MSD Still Alive And Prospering!

In Volume 7, Issue 01's NEWS BRK column, we reported the demise of Micro System Development, better known as MSD. Well, a Mr. Allen Dermody sent us a letter pointing at that the company is still doing well, as he had his MSD drive recently serviced by them. It seems that they are no longer developing disk drives for Commodore machines, but still exist as a company, doing mainly

point of sale cash registers, and apparently doing quite well. MSD may be reached at:

MSD Systems, Inc.
10031 Monroe Drive Suite 206
Dallas, Texas 75229
(214) 357-8587

The address for the MSD Information Exchange is:

Paul Eckler
2705 Hulman Street
Terre Haute, IN
47803

Twin Cities 128: The Commodore 128 Journal

As quoted from the April, 1986 edition; Volume 1, Issue 4

" 1571 Music Critic: While doing some simple file transfers to disk while listening to ZZ-Top's Afterburner album, I noticed my 1571 was having problems saving. The green light was furiously winking on and off in dazed distress, and the read/write head was buzzing loudly in a vain attempt to find the proper place on the disk to do its job."

If you had read through the balance of this article, you would discover that the Commodore 1571 drive dislikes the music of ZZ Top (the kick-drums knock the drive out of alignment) but loves Stevie Nicks; that is, according to The Abominog's Ruminations column. Although a little bit on the weird side at times, reading through this journal is a delight to be enjoyed by all.

What better way to tell of a journal's contents than to list it - The Table Of Contents:

Rumors, Opinions, Mayhem
Abominog's Ruminations
Sid Vicious Bytes
Print Using Explained
Intro To Music 7.0
CP/M Update
Printer Files Explained
Commodore Experts Speak
Sparrow's Slick Tips

... plus reviews on Paperback Filer, Commodore 1350 Mouse, Commodore 1670 Modem, Fleet System 3/128, Matrix, Leader Board and Project: Space Station

As stated all along, a journal not to be without. $1.75 (US funds) per issue or $22.00 for a year's subscription. Available from:

Twin Cities 128
1607 Hewitt Avenue, Suite 4
PO Box 4526
Saint Paul, MN 55104

Creative Writer

"Although the computer was quite elegantly constructed, it was not as sleek as Sharon's calf." The preceding sentence was not written on a computer: it was written by a computer. CREATIVE WRITER is not a word-processor. It is a writer, a writer who never gets writer's block. And it is one of the most unusual and amusing and educational programs ever written for the C-64.

The program was written by a Canadian poet and novelist for use in teaching creative-writing in Ontario schools, where both students and teachers have invariably found it both fascinating and hilarious. While extremely easy to use, C.W. is quite capable of extremely complex linguistic investigation. This is because of special modules which generate individualized vocabulary and syntax files which can be utilized by the main program. Th flexibility of C.W. makes it possible to do everything from analysis and simulation of established authors to curse-generation tailored to abuse your 'favourite' politicians and friends.

The randomly generated sentences are infinitely variable in terms of vocabulary and sentence structure, and they are grammatically correct. The products of a 'creative writing session' can be saved to disk as a standard sequential file (for later editing with a word-processor) or streamed to a printer. While we can't promise you that C.W. will write the Great Canadian Novel for you, we can promise that it will be one of the most amusing and unusual pieces of software in your collection!

Creative Writer has a retail price of $29.95. To order, phone or send a cheque or money-order to:

The G.A.S.S. Company
970 Copeland
North Bay, Ontario, Canada
P1B-3E4 (705) 474-9602

Sector Surgeon For The C-64

Sector Surgeon is the first advanced disk repair program for the C-64, SX-64 and C-128.

Unlike its predecessors, Sector Surgeon is written totally in machine language. Sector Surgeon uses FDC disk routines instead of the less versatile IP routines to enhance its flexibility. This also allows writing and reading of protected disks from tracks 1 to 35. Also, the block availability map sector need not be intact for Sector Surgeon to work. Instructions are available at the touch of a key.

The sector information is displayed in a window in screen code and the byte under the cursor is displayed simultaneously in (1) screen code (2) decimal (3) hex (4) binary (5) basic. All header information and data block checksums can be displayed in decimal, hex and screen codes. Shows actual track number and displays the forward link.

Sector Surgeon will read information under almost all errors and will write information underneath header errors. Error 23 will be corrected automatically upon writing to a disk. You can copy sectors even to another disk.

Sector Surgeon has been tested and found to work with the 1541, VIC-1541, 1540, 1541 Flash! and SX-Flash! For more information contact:

Bak Room Boys Software
2306 N MacArthur Blvd.
Oklahoma City, OK 73127
(405) 946-2888

Three MIDI Data Storage Programs For The Commodore 64

The CZ Dumpstor is a patch librarian for the Casio CZ101, CZ1000, CZ3000 and CZ5000 synthesizers. Three banks of sixteen patchs can be in memory at one time, and the program includes 128 professional patches. The CZ Dumstor sells for $54.95 (US funds).

The Data Dumpstor is a very powerful data storage program that stores patches, sequences, drum patterns and other MIDI information from over 20 different instruments. Data is accepted from the DX-7, RX-11, TX-7, DX-9, QX-7, and the new DX-100, and several more Yamaha instruments. Data from instruments by Korg, Sequential, Oberheim and others can also be stored and retrieved. It holds 36k and MIDI info from several instruments can be stored or sent at the same time. The Data Dumpstor sells for $59.95 (US funds).

The TR-707 Dumpstor accepts drum patterns and songs from the Roland TR-707 and TR-727 drum machines. It eliminates the need for cassette data

storage, and also uses the ultra-fast disk MIDI file loading routine used by the CZ Dumpstor and the Data Dumpstor. It sells for $39.95 (US funds). For more information, contact:

Music Service Software
801 Wheeler Road
Madison, WI 53704

The Electronic Shoe Box Accounting Systems For The C64

Dating back to its original release in 1983 for use with the Pet/CBM microcomputers, the new Shoe Box II Accounting and Shoe Box Payroll systems for the Commodore 64 and C128 have arrived. Two years of unrelenting work has gone into making the Shoe Box systems the right choice for all your accounting needs. While retaining complete compatibility with the previous Shoe Box data structures, the speed of program execution has now been further increased by a factor of six times. Copy protection is employed with this package, but the "dongle" method of protection has been chosen to allow duplicates of program and data files without problems. The manuals supplied are well written, well presented and very well bound. From all aspects, the new Shoe Box accounting systems have the markings of a truly good system, one that may be just right for your accounting needs.

Take a look at the fine features to be enjoyed in the use of this system:

- easy initial set-up (20 minutes).
- calculates tax, CCA and prorated expenses automatically.
- automatic posting to General Ledger.
- automatic addressing of invoices and customer statements.
- print and sort by 5 digit code.
- prints mailing labels.
- supports most Commodore and compatible disk drives and printers.
- 97 General Ledger accounts.
- income statement and balance sheet.
- Canadian half-year rule for CCA.
- 200 "Aged" customer sub-accounts.
- 70 data journals per disk with up to 13,000 journal entries.
- 9 million dollar account limit.

Plus much, much more. The Shoe Box Canadian Payroll package has a retail price of $79.95 (Canadian funds). The Shoe Box II Accounting system has a retail price of $89.95 (Canadian funds). For more information, contact:

John Dunlop and Associates Ltd.
RR Number 5
Orangeville, Ontario, Canada
L9W-2Z2 (519) 941-9572

Freedom Assembler-128 For The Commodore 128

The Freedom Assembler-128 is a symbolic assembler, written entirely in assembly language, by a professional computer programmer. Now, you too can write programs for the 6502/6510/65C02/8502 family of microcomputers. With this assembler, programming is faster and easier than ever. The Freedom Assembler-128 takes full advantage of the C-128's lighting speed and super memory capacity. No more having to link too long files, no more long waits for your disk assembler to load. The Freedom Assembler-128 is always ready when you need it. The cartridge format allows it to remain plugged into your computer until it is needed, with no interference with your other programs. The Freedom Assembler-128 works in both 40 and 80 column mode.

The Freedom Assembler-128 is available for $49.95 (US funds). Postage is paid by Hughes Associates. Order from:

Hughes Associates
45341 Harmony Lane
Belleville, MI 48111
(313) 699-1931

Rebel Assembler/Editor For The Commodore 64 and 128

A new dawn is on the horizon for Commodore programmers who program in the native machine language of either the Commodore 64 or Commodore 128 PC.

A sophisticated 65XX/85XX assembler that assembles source code at unbelievably lighting speed, Rebel is integrated in both the 64 and 128 versions with an extremely versatile and helpful screen editor. The 128 editor in the 80 column mode even provides an on-screen help menu and true split-screen editing. Rebel takes no functions away from the user, giving total access to the computer's powerful Basic language at all times, while providing extra facilities such as a file lister, list freezer, search and replace functions and much more. The 64 version, because of the limitations of Basic 2.0, adds other features such as fast renumber, page flipping, hexadecimal - decimal - ASCII conversions, list scrolling and more.

A nice thing about Rebel is that it is available now, at a suggested retail price of only $29.95. For more information, please contact:

Nu-Age Software
2311 28th Street North
St. Petersburg, FL 33713
(813) 323-8389

Liz Deal's Basic Program Converter

List/conv by Elizabeth Deal is a program which converts programs from one Commodore machine into a format such that they can be listed and edited on another machine. If you don't know which machine a particular program comes from, List/conv will do its best to figure it out for you. The newly created program may not directly run on the target machine, but at least the keyword tokens won't be messed up; they will be converted to equivalent tokens in the new machine, or actual ASCII strings where no such keywords exist.

Included on the disk with Liz's List/conv at no charge is Jim Butterfield's public-domain "lister" program, which runs a bit slower than the machine language List/conv, but tends to handle control characters better. The disk with both programs is only $10.00 U.S., available from:

Liz Deal
337 West 1st Ave
Malvern, PA, 19355

10 and 20 Megabyte Hard Disk Drives For The C64

Fiscal Information Services have broken the speed barrier with the first really fast hard disk system for the Commodore 64. How fast is really fast. It's fast enough to Load a full screen of high-resolution color graphics (about 11 kbytes) in less than one second! In fact, it is up to 43 times faster than Commodore's standard 1541 floppy disk drive.

FIS's Lt. Kernal disk drive carries an on-board DOS. It's a substantial upgrade to C64's Basic since it adds run-time functions and several CP/M like command line function. Lt. Kernal interfaces via the expansion/cartridge port and transparently implements all Commodore 1541 DOS functions.

The capacity of 10 or 20 MB is standard; larger capacities and streaming tape backup are available as extra-cost options. The price of the 10 MB Lt. Kernal is $1,595.00 (US). For more information:

Fiscal Information Inc.
143 Executive Circle
Daytona Beach, FL 32014
(904) 253-6222

Quick Brown Box:
An 8k Read/Write Cartridge For The Commodore 64

The Quick Brown Box is a battery-backed-up RAM cartridge for the C-64 (or C-128 in 64 mode) that allows the Basic or machine language programmer to load or write programs and save them for immediate availability at power up. For the Expansion Port, up to 8k of frequently used utilities, wedges, or games may be stored, using no hardware besides the computer itself. Includes Write Protect switch and Reset button, and is shipped with Auto-Start, Basic utilities and M.L. Monitor. USA $39.00 plus $3.00 Handling and Mailing, Canadian $50.00 plus $4.00 H&M. Cheques accepted.

Brown Boxes, Inc.
26 Concord Road
Bedford, MA 01730
(617) 275-0090

1540/1541 Drive Alignment System By SchuLace Enterprises

As good as they are, in normal usage, the 1540/1541 Disk Drives are subject to knocking themselves out of alignment. It happens enough to warrant the attention of all users. SchuLace uses drive electronics coupled with a pair of innovative indicators to correct the problem. We economically show you how to get your problem drive reading those early-recorded and commercial diskettes. As the name implies, it will get you back On-Track. The On-Track System is the product of nearly two years research, development and improvement. It's multi-functions and advantages will prove its immeasurable value with use. On-Track has a suggested retail price of $15.98 (US funds) plus $2.00 P&H. For more information, contact:

SchuLace Enterprises
P.O. Box 771
Cascade, Maryland 21719

Astrology Program For Commodore 64

This program will run on a Commodore 64 system with a disk drive and printer (optional). Choose between TROPICAL or SIDEREAL zodiac. It is based on a unique ancient eastern system of astrology using divisions of zodiac into 12 signs, 27 asterisms, 108 navamsa quarters and 249 sub-asterismal divisions. Uses a unique lunar progression (vimsottary dasa). It also has a time-tested horary chart option for drawing charts for answering questions or when the birth time is unknown. Longitudes of 9 planets and 2 lunar nodes, their positions, strengths, aspects and planetary periods, etc., are displayed on-screen. The screen can be dumped to a connected printer (device #4). In addition, a separate **formatted hardcopy** option supports 1526/MPS-802 and Epson Standard Code compatible printers. A quick horoscope snapshot option and directory display are supported. Documentation for operating the program and a treatise outlining the predictive aspects (with important reference material and example horoscope analysis) are included for those wishing to learn the system. Backup disk included. Available for $50.00 (Canadian funds) from:

ROHINI
P.O. Box 9
St. Norbert Post Office
Winnipeg, Manitoba, Canada
R3V-1L5

Multiplex Eight RS232 Ports Onto A Single X.25 Line

New from Black Box Corporation. X.25 PAD/Concentrator lowers cluster access costs. The use of X.25 packet data networks (such as DataPac) has grown rapidly as users recognise PDN's cost-saving advantage over leased lines. Now, get even greater use out of a single X.25 access line with Black Box's X.25 PAD/Concentrator. This device has an X.25 "in" connector and 8 bi-directional RS232 female ports. It permits up to eight terminals or computer ports to multiplex onto a single X.25 line, making it a cost effective way to connect a cluster of remote asynchronous terminals to a non X.25 host, or provide multiple remote terminal access to an X.25 host computer.

Easy to install, maintain and troubleshoot, X.25 PAD has a powerful, user friendly command facility which provides a comprehensive set of configuration and control functions, such as long call forms, abbreviated call and autocall, class selection, channel calling restrictions and incoming call validation. It can support direct, dedicated and dial up terminal connections at speeds up to 9600 bps. The unit is fully compatible with CCITT, X.3, X.28, X.29, X.121 and X.25 access lines.

For a free catalog that describes this and 500 other data communications and computer devices available from Black Box, write:

X.25 PAD/Concentrator
Black Box Corp.
P.O. Box 12800
Pittsburgh, PA
15241 (412) 746-5500

/SPEEDPAK/ Speedscript Enhancer

Upstart Publishing proudly presents /SPEEDPAK/, the C64 Speedscript 3.0-3.2 enhancer. /SPEEDPAK/ adds six new commands, three printer codes, and eight user-definable 31-character macro phrase keys to Speedscript. Among the new features are:

- Alternate screens: Edit, cut, and copy between documents instantly
- Help screen installer: use one built-in help screen at a time. Four free samples are included, or you can create your own.
- SCREEN font installer: four special character sets are included for onscreen viewing (these do not print to the page).
- File encryption: For security, scramble your text files before saving and recover them after loading (using your own 32-character code).
- Code conversion: Convert screen codes to Commodore ASCII and vice versa.
- Defaults: Set disk or tape as default storage device, and set a standard printing device and secondary address.
- Dvorak keyboard option: Use the world's fastest keyboard arrangement - including a special help screen for beginners.

Three additional printer codes work with the alternate screen feature to provide a RAM-based for letter mail merge, allowing you to merge a record into the form letter, or skip forward or backward through records - all with no disk access during printing.

Best of all, you can save Speedscript, /SPEEDPAK/, a character set, a help screen, and preset defaults (including background and border colours) as one easy to load bundle! And the /SPEEDPAK/ disk comes with printed instructions, three disk-baséd tutorials, and three sample files that show you how to use /SPEEDPAK/'s handy functions. Price is $15.00 US, $12.00 for quantities of ten or more. Order from:

Upstart Publishing Dept. TN
P.O. Box 22022
Greensboro, NC 27420

Speed-Plus Speedscript Enhancer

The Speed-Plus Speedscript enhancer from Lidon Enterprises adds the following features to Speedscript 3.0-3.2: Justification, Tabs, 2 column/2 side printing, word wrap toggle, preview of output to screen, printing of any section of a document, user-defined printer commands, and change of printer secondary address "on the fly". Price is $24.95 US including P & H, from:

Lidon Enterprises Dept. UP
P.O. Box 773
Elm Grove, WI 53122

JOIN TPUG

The largest Commodore Users Group

Benefit from:

Access to library of public domain software
for C-64, VIC 20 and PET/CBM

Magazine (10 per year) with advice from

Jim Butterfield
Brad Bjomdahl
Liz Deal

TPUG yearly memberships:

Regular member (attends meetings)	—$35.00 Cdn.
Student member (full-time, attends meetings)	—$25.00 Cdn.
Associate (Canada)	—$25.00 Cdn.
Associate (U.S.A.)	—$25.00 U.S.
	—$30.00 Cdn.
Associate (Overseas — sea mail)	—$35.00 U.S.
Associate (Overseas — air mail)	—$45.00 U.S.

FOR FURTHER INFORMATION:
Send $1.00 for an information catalogue
(tell us which machine you use!)

To: TPUG INC.
DEPT. A,
101 DUNCAN MILL RD., SUITE G7
DON MILLS, ONTARIO
CANADA M3B 1Z3

COMAL INFO
If you have COMAL— We have INFORMATION.

BOOKS:
- COMAL From A To Z, $6.95
- COMAL Workbook, $6.95
- Commodore 64 Graphics With COMAL, $14.95
- COMAL Handbook, $18.95
- Beginning COMAL, $22.95
- Structured Programming With COMAL, $26.95
- Foundations With COMAL, $19.95
- Cartridge Graphics and Sound, $9.95
- Captain COMAL Gets Organized, $19.95
- Graphics Primer, $19.95
- COMAL 2.0 Packages, $19.95
- Library of Functions and Procedures, $19.95

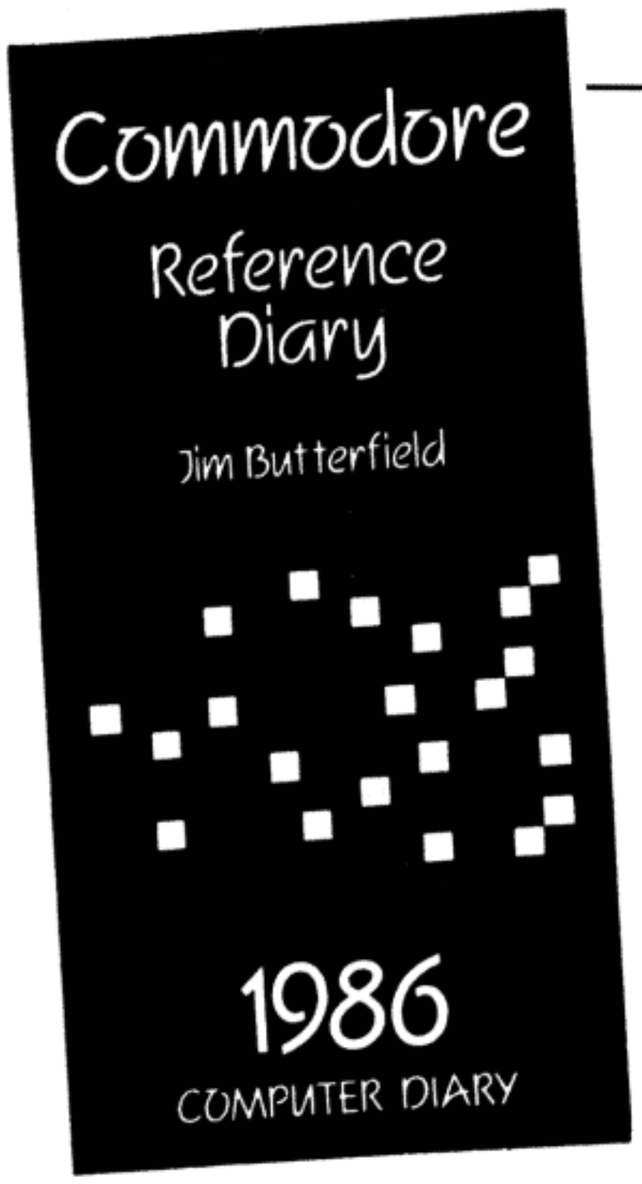
OTHER:
- COMAL TODAY subscription, 6 issues, $14.95
- COMAL 0.14, Cheatsheet Keyboard Overlay, $3.95
- COMAL Starter Kit (3 disks, 1 book), $29.95
- 19 Different COMAL Disks only $94.05
- Deluxe COMAL Cartridge Package, $128.95 (includes 2 books, 2 disks, and cartridge)

ORDER NOW:

Call TOLL-FREE: 1-800-356-5324 ext 1307 VISA or MasterCard ORDERS ONLY. Questions and Information must call our Info Line: 608-222-4432. All orders prepaid only—no C.O.D. Add $2 per book shipping. Send a SASE for FREE Info Package or send check or money order in US Dollars to:

COMAL USERS GROUP, U.S.A., LIMITED
5501 Groveland Ter., Madison, WI 53716

TRADEMARKS: Commodore 64 of Commodore Electronics Ltd.; Captain COMAL of COMAL Users Group, U.S.A., Ltd.



From The Guru Himself!
The 1986 Commodore Reference Diary

A 65 page reference section that includes:
- All hardware specifications including the C128 and PC10/20
- Useful memory locations
- Useful programs
- SuperCharts
- BASIC and machine language hints
- Hexadecimal conversion
- Sound, video
- and more

The full calendar and date book includes:
- National holidays in ten countries
- Personal notes
- 1987 forward planner
- Name, address, telephone section

Just $5.95
(plus 50¢ postage and handling)

Order Your Copy Today!

Canada
The Transactor
500 Steeles Avenue
Milton, Ontario
L9T 3P7

USA
The Transactor
277 Linwood Avenue
Buffalo, New York
14209

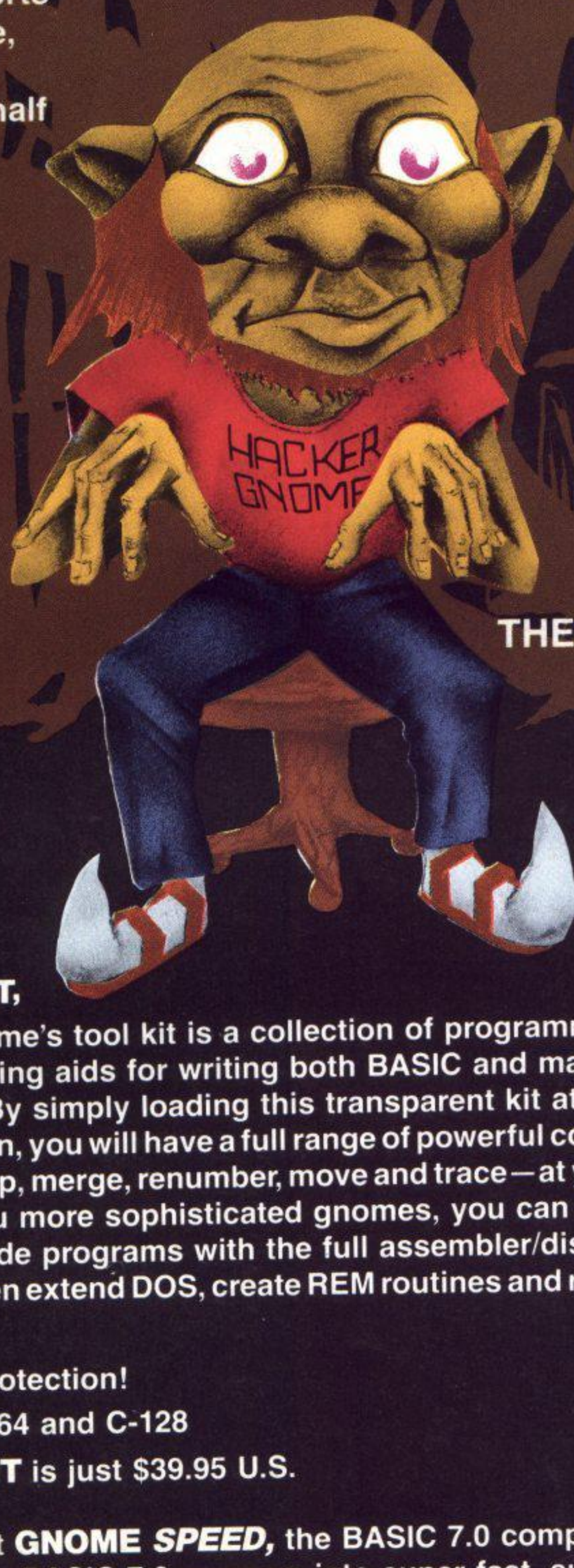
Dealer Orders:

Canada
Norland Agencies
251 Nipissing Road
Milton, Ontario
L9T 4T7
(416) 876 - 4774

USA
MicroPace Distributing
1510 North Neil Street
Champaign, Illinois
61820
1 800 362-9653

After hours and hours of staring at that little screen
and typing at remarkable speed
he went GNOME

But thanks to the efforts
of this Hacker Gnome,
you can cut your
programming time in half
and triple your
productivity!



Introducing

GNOME KIT

THE PROGRAMMING TOOL KIT

GNOME KIT,
Hacker Gnome's tool kit is a collection of programming, designing and debugging aids for writing both BASIC and machine code programs. By simply loading this transparent kit at the start of your work session, you will have a full range of powerful commands—such as find, dump, merge, renumber, move and trace—at your gnometips. And, for you more sophisticated gnomes, you can easily write machine code programs with the full assembler/disassembler and editor – Even extend DOS, create REM routines and restore corrupted disks.

No Copy Protection!

For both C-64 and C-128

GNOME KIT is just $39.95 U.S.

Don't forget **GNOME *SPEED,*** the BASIC 7.0 compiler. Transform virtually any BASIC 7.0 program into super fast, super compact machine code. Only $59.95.

U.S. Mail Orders:
SM Software, Inc.
P.O. Box 129
Kutztown, PA 19530
(215) 683-5699

Canadian Mail Orders:
The Transactor
(416) 878-8438

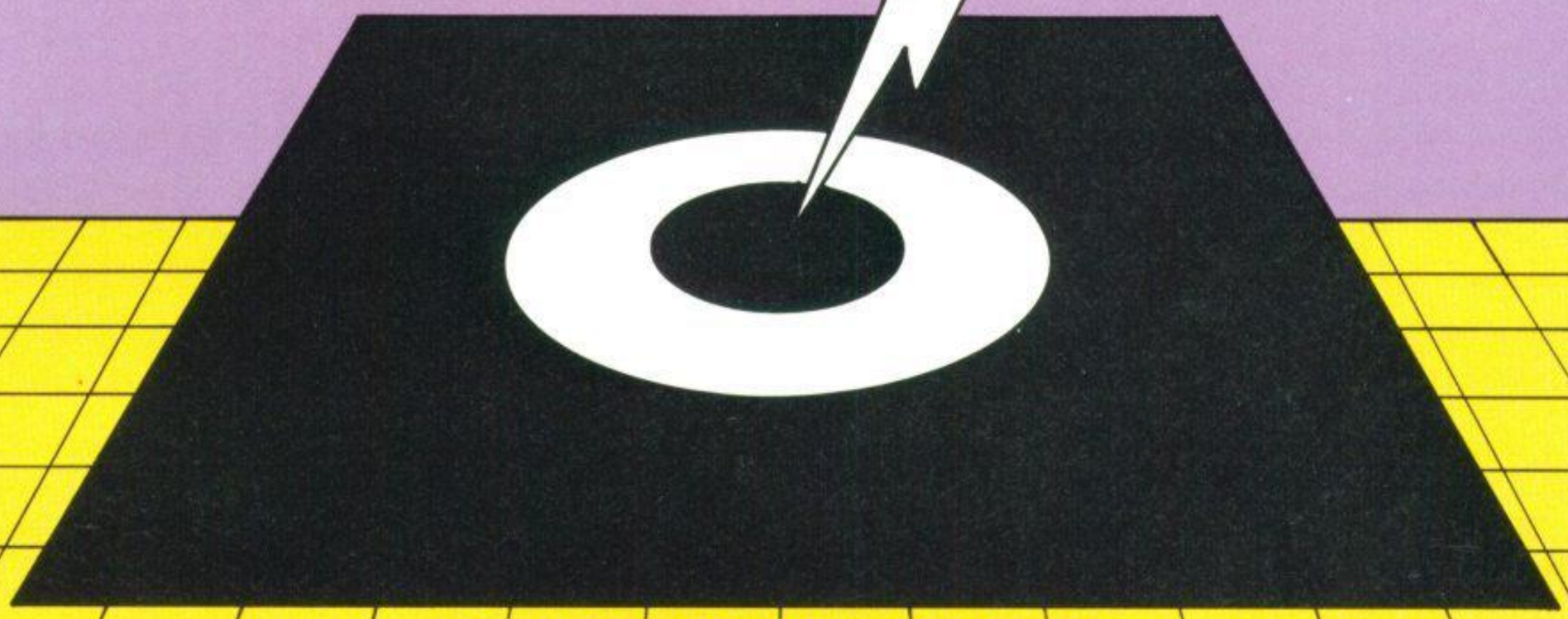
Dealer Inquiries:
Micro-Pace, Inc.
(217) 356-1884

See Order Card

INTRODUCING

SUPER KIT/1541

Has it all!



BY MARTY FRANZ & JOE PETER

SINGLE/DUAL NORMAL COPIER
Copies a disk with no errors in 32.68 seconds. dual version has graphics & music.

SINGLE/DUAL NIBBLE COPIER
Nibble Copies a disk in 34.92 seconds. Dual version has graphics & music.

SINGLE/DUAL FILE COPIER
7 times normal DOS speed. Includes multi-copy, multi-scratch, view/edit BAM, & NEW SUPER DOS MODE. In Super DOS Mode, it transfers 7-15 times normal speed, copies 150 blocks in 23 seconds.

TRACK & SECTOR EDITOR
Full editing of t&s in hex, dec, ascii, bin. Includes monitor/disassembler with printout commands.

GCR EDITOR
Yes disk fans, a full blown sector by sector or track by track GCR Editor. Includes TRUE Bit Density/Track Scan.

3 SUPER DOS FAST LOADERS
Over 15 times normal DOS speed. Super DOS Files are still Commodore DOS compatible. Imagine loading 150 blocks in 10 seconds.

SUPER NIBBLER/ SUPER DISK SURGEON
Quite frankly, these will provide you the user with the backup you need! Even copies itself.

$29.95 U.S.

PLUS $3.00 SHIPPING/HANDLING CHARGE — $5.00 C.O.D. CHARGE

PRISM SOFTWARE

401 LAKE AIR DR., SUITE D • WACO, TEXAS 76710
ORDERS (817) 757-4031 • TECH (817) 751-0200
MASTERCARD & VISA ACCEPTED

SUPER KIT/1541 is for archival use only! We do not condone nor encourage piracy of any kind.

See center page for mail order card.

THE TIME SAVER

Type in a lot of Transactor programs?
Does the above time and appearance of the sky look familiar?
With The Transactor Disk, any program is just a LOAD away!

Only $7.95 Per Issue
6 Disk Subscription (one year)
Just $45.00
(see order form at center fold)